

UNCLASSIFIED



Australian Government

Department of Defence

Defence Science and
Technology Organisation

Implementation of Geometric Algebra in MATLAB[®] with Applications

Leonid K. Antanovskii

Weapons and Combat Systems Division

Defence Science and Technology Organisation

DSTO-TR-3021

ABSTRACT

Geometric Algebra is the most appropriate unifying mathematical language to describe diverse problems in mathematics, physics, engineering and computer science. In combination with *Projective Geometry* it provides an efficient framework for computer vision and robotics, where image processing and recognition play the central rôle. This document addresses a gentle introduction to *Geometric Algebra* followed by its implementation in MATLAB. The developed fully vectorized code is thoroughly tested. Several applications are presented in the form of unit tests, amongst which are some basic algorithms for the reconstruction of a three-dimensional structure from two-dimensional images.

APPROVED FOR PUBLIC RELEASE

UNCLASSIFIED

Published by

DSTO Defence Science and Technology Organisation

PO Box 1500

Edinburgh, South Australia 5111, Australia

Telephone: 1300 333 362

Facsimile: (08) 7389 6567

© Commonwealth of Australia 2014

AR No. AR 016-076

September, 2014

APPROVED FOR PUBLIC RELEASE

Implementation of Geometric Algebra in MATLAB[®] with Applications

Executive Summary

Geometric Algebra is the most appropriate unifying mathematical language to describe diverse problems in mathematics, physics, engineering and computer science. In combination with *Projective Geometry* it provides an efficient framework for computer vision and robotics, where image processing and recognition play the central rôle. This document addresses a gentle introduction to *Geometric Algebra* followed by its implementation in MATLAB. The developed fully vectorized code is thoroughly tested. Several applications are presented in the form of unit tests, amongst which are some basic algorithms for the reconstruction of a three-dimensional structure from two-dimensional images.

THIS PAGE IS INTENTIONALLY BLANK

Contents

1	Introduction	1
2	Geometric algebra	1
2.1	Clifford algebra of 3-dimensional vector space	6
2.2	Clifford algebra of 4-dimensional vector space	7
3	Projective geometry	8
3.1	Incidence relations in projective plane \mathbb{P}^2	9
3.2	Incidence relations in projective space \mathbb{P}^3	10
4	Application to 3D reconstruction	12
5	Description of the developed MATLAB code	15
6	Discussion	16
	References	17

Appendices

A	Multiplication tables	18
B	Listing of clifford_algebra.m	21
C	Listing of clifford_algebra_test.m	34
D	Listing of geometric_algebra_test.m	42

Figures

1	Pinhole camera	12
2	Epipolar geometry	13

Tables

A1	Geometric products of the basis multivectors of $\mathcal{C}(\mathbb{R}^3)$	18
A2	Progressive outer products of the basis multivectors of $\mathcal{C}(\mathbb{R}^3)$	18
A3	Regressive outer products of the basis multivectors of $\mathcal{C}(\mathbb{R}^3)$	18
A4	Geometric products of the basis multivectors of $\mathcal{C}(\mathbb{R}^4)$	19
A5	Progressive outer products of the basis multivectors of $\mathcal{C}(\mathbb{R}^4)$	20

THIS PAGE IS INTENTIONALLY BLANK

1 Introduction

It has been recognized that *Geometric Algebra* is the most appropriate unifying mathematical language to describe diverse problems in mathematics, physics, engineering and computer science [Hestenes & Sobczyk 1987, Corrochano & Sobczyk 2001, Perwass 2009]. In particular, in combination with *Projective Geometry* it provides an efficient framework for computer vision and robotics, where image processing and recognition play the central rôle.

There are several algebraic structures suitable for the formalization of geometric operations, such as Gibbs vector algebra, Grassmann algebra, Grassmann–Cayley algebra and Clifford algebra. The classical vector algebra operates on vectors but has limitations when dealing with more complex geometric constructs. Grassmann algebra is equipped with an exterior product, also called a progressive outer product, which is useful for building higher dimensional objects from lower dimensional objects, for example, when creating a line joining two points or a plane passing through three points in a projective space. Grassmann–Cayley algebra is additionally equipped with a regressive outer product which formalizes the operation of intersection of linear objects, such as the creation of a line of intersection between two planes. Finally, Clifford algebra additionally contains an inner product which is used to describe parallelism and orthogonality.

In this document we confine ourselves to Clifford algebra generated by a vector space equipped with a positive definite scalar product. Note that the Grassmann and Grassmann–Cayley algebras can be factored out from Clifford algebra, and hence they are part of it. In our context these algebraic structures are identified with *Geometric Algebra*.

We provide a gentle introduction to *Geometric Algebra* and *Projective Geometry* to fix definitions and notations, and recall basic relations between algebraic and geometric operations. The developed MATLAB code for a range of Clifford algebras, based on the algebra multiplication tables given in Appendix A, is presented in Appendix B. The fully vectorized code is thoroughly tested against the basic properties of the Clifford algebra operations and derived identities by a suite of unit tests given in Appendix C.

When combined with projective spaces, the real power of Clifford algebra emerges. In particular, the effectiveness of *Geometric Algebra* manifests itself by an elegant and concise form of equations for intricate geometric constructs. We present another suite of unit tests in Appendix D which demonstrates this power. We apply this technique to the derivation of basic algorithms for the projection of a three-dimensional object to two-dimensional images, followed by its reconstruction from a sequence of images.

2 Geometric algebra

Consider an n -dimensional vector space \mathbb{R}^n over the field of real numbers \mathbb{R} , equipped with a positive definite scalar product denoted by the centred dot

$$\mathbb{R}^n \times \mathbb{R}^n \ni (a, b) \mapsto a \cdot b \in \mathbb{R}. \quad (1)$$

Choose an orthonormal basis $\{e_1, \dots, e_n\}$ characterized by the identities $e_i \cdot e_j = \delta_{ij}$ where δ_{ij} is the Kronecker symbol. The Clifford algebra $\mathcal{C}(\mathbb{R}^n)$ is defined by the following

multiplication rules on the basis elements

$$e_i e_j + e_j e_i = 2\delta_{ij} \quad (2)$$

in combination with the additional axioms of associativity and distributivity [Lang 2002, Garling 2011, Hestenes & Sobczyk 1987].

We denote the multiplication operation in the Clifford algebra by juxtaposition. This product is called the geometric or Clifford product to distinguish it from other products to be introduced. The multiplication rules on the basis vectors can be written as

$$e_i e_j = e_i \cdot e_j + e_i \wedge e_j \quad (3)$$

where the inner (scalar) and outer (wedge) products are expressed for any vectors a, b in terms of the geometric product

$$a \cdot b = \frac{1}{2} (a b + b a) , \quad a \wedge b = \frac{1}{2} (a b - b a) . \quad (4)$$

For the basis orthonormal vectors we have $e_i^2 = 1$ and $e_i e_j = e_i \wedge e_j = -e_j \wedge e_i = -e_j e_i$ provided that $i \neq j$. The elements of $\mathcal{C}(\mathbb{R}^n)$ are called multivectors, hypercomplex numbers or Clifford numbers.

The basis multivectors of the Clifford algebra are constructed by the collection $\{e_I\}$ where the compound index I is a subset of the set $\{1, 2, \dots, n\}$. For convenience we adopt the notation $e_\emptyset = 1$ (when $I = \emptyset$) and set

$$e_I = e_{i_1 i_2 \dots i_k} \equiv e_{i_1} e_{i_2} \dots e_{i_k} \quad \text{for } I = \{i_1, i_2, \dots, i_k\} \quad (5)$$

where we will always assume that $1 \leq i_1 < i_2 < \dots < i_k \leq n$. Obviously, the basis multivectors contain the basis vectors when $k = 1$.

It is straightforward to generate the multiplication table of the basis multivectors of the Clifford algebra, because $e_I e_J = \pm e_K$ for some compound index K . The procedure is as follows. Move each factor e_{j_q} for $q = 1, \dots, l$ in the product $e_{i_1} e_{i_2} \dots e_{i_k} e_{j_1} e_{j_2} \dots e_{j_l}$ to the left by swapping it with the immediate left neighbour e_{i_p} while $i_p > j_q$, and changing the sign of the product according to the anti-commutativity law. Stop when either $i_p < j_q$ or $i_p = j_q$. In the latter case set $e_{i_p} e_{j_q} = 1$. In the end of the procedure the indices of the product factors become sorted which produce the final index K and the sign of the equivalent product. For example,

$$e_{23} e_{12} = e_2 e_3 e_1 e_2 = -e_2 e_1 e_3 e_2 = e_1 e_2 e_3 e_2 = -e_1 e_2 e_2 e_3 = -e_1 e_3 = -e_{13} . \quad (6)$$

In particular, we have proved that a general multivector A is expressed by the formal sum

$$A = \sum_I A_I e_I \quad (7)$$

where A_I are real numbers.

The multiplication rule is extended to all multivectors by linearity using the distributivity law. By virtue of the formal sum (7), we immediately arrive at the direct sum decomposition of the Clifford algebra

$$\mathcal{C}(\mathbb{R}^n) = \mathcal{C}_0(\mathbb{R}^n) \oplus \mathcal{C}_1(\mathbb{R}^n) \oplus \dots \oplus \mathcal{C}_n(\mathbb{R}^n) \quad (8)$$

where $\mathcal{C}_k(\mathbb{R}^n)$ is spanned by those e_I whose index I is composed of k integers. The number k is called the grade of the vector subspace $\mathcal{C}_k(\mathbb{R}^n)$, and the elements of $\mathcal{C}_k(\mathbb{R}^n)$ are called k -vectors. The dimension of $\mathcal{C}_k(\mathbb{R}^n)$ is equal to ‘ n choose k ’, namely

$$\dim \mathcal{C}_k(\mathbb{R}^n) = \binom{n}{k} \equiv \frac{n!}{k!(n-k)!}. \quad (9)$$

In particular,

$$\dim \mathcal{C}(\mathbb{R}^n) = \sum_{k=0}^n \dim \mathcal{C}_k(\mathbb{R}^n) = \sum_{k=0}^n \binom{n}{k} = 2^n. \quad (10)$$

Hence the Clifford algebra generated by an n -dimensional vector space is 2^n -dimensional.

The vector subspace $\mathcal{C}_0(\mathbb{R}^n)$ is isomorphic to \mathbb{R} , which is also a central subalgebra. By definition, a subalgebra is central if all its elements commute with the elements of the algebra [Lang 2002]. The centre of an algebra is the set of elements commuting with all elements, which is obviously a (maximal) central subalgebra [Lang 2002]. It is known that the centre of $\mathcal{C}(\mathbb{R}^n)$ is $\mathcal{C}_0(\mathbb{R}^n)$ for n even, or $\mathcal{C}_0(\mathbb{R}^n) \oplus \mathcal{C}_n(\mathbb{R}^n)$ for n odd.

Note that 0-vectors are called scalars, n -vectors are pseudoscalars, and 1-vectors $\mathcal{C}_1(\mathbb{R}^n)$ are naturally identified with vectors \mathbb{R}^n . The following direct sum of the even-grade subspaces

$$\mathcal{C}^+(\mathbb{R}^n) = \mathcal{C}_0(\mathbb{R}^n) \oplus \mathcal{C}_2(\mathbb{R}^n) \oplus \dots \oplus \mathcal{C}_{2[n/2]}(\mathbb{R}^n) \quad (11)$$

is in fact a subalgebra because it is closed under multiplication. Indeed, the product of two even-grade elements has an even grade. The dimension of $\mathcal{C}^+(\mathbb{R}^n)$ is equal to 2^{n-1} . The even subalgebra $\mathcal{C}^+(\mathbb{R}^n)$ plays an important rôle in the definition of spinors [Hestenes & Sobczyk 1987, Hurley & Vandyck 2000].

It is worthwhile noting that, since scalars are embedded in a Clifford algebra, the multiplication by scalars coincides with the ordinary multiplication of algebra elements. Therefore, from the algebraic point of view, Clifford algebras are indeed rings [Lang 2002].

Let us extend the inner and outer products from 1-vectors to all multivectors. A general multivector A has the direct sum decomposition

$$A = \sum_{k=0}^n \langle A \rangle_k \quad (12)$$

where the grade-projection operator of $\mathcal{C}(\mathbb{R}^n)$ to $\mathcal{C}_k(\mathbb{R}^n)$ is denoted by the angular bracket

$$\mathcal{C}(\mathbb{R}^n) \ni A \mapsto \langle A \rangle_k \in \mathcal{C}_k(\mathbb{R}^n). \quad (13)$$

It is convenient to extend the grade-projection operator to identical zero for $k > n$ or $k < 0$. It suffices to define the inner and outer products on elements of given grades and extend them by linearity to all multivectors. Having said that we define

$$A \cdot B = \sum_{k=0}^n \sum_{l=0}^n \langle \langle A \rangle_k \langle B \rangle_l \rangle_{|k-l|}, \quad (14)$$

$$A \wedge B = \sum_{k=0}^n \sum_{l=0}^n \langle \langle A \rangle_k \langle B \rangle_l \rangle_{k+l} . \quad (15)$$

It is easy to verify that these definitions are consistent with the original definitions for 1-vectors. Note that $\alpha \cdot A = A \cdot \alpha = \alpha \wedge A = A \wedge \alpha = \alpha A = A \alpha$ for any scalar $\alpha \in \mathcal{C}_0(\mathbb{R}^n)$. The inner product combines the left and right contraction in a single expression. It is worthwhile emphasizing that neither is the inner product symmetric nor is the outer product antisymmetric. It is important to stress that the outer product is associative, whereas the inner product is not.

The following identities exploiting the outer and inner products can be proved in a straightforward manner

$$a \cdot (b \wedge c) = (a \cdot b) c - (a \cdot c) b \quad \forall a, b, c \in \mathbb{R}^n , \quad (16)$$

$$(a \wedge b) \cdot (c \wedge d) = (a \cdot d) (b \cdot c) - (a \cdot c) (b \cdot d) \quad \forall a, b, c, d \in \mathbb{R}^n , \quad (17)$$

$$a \wedge C \wedge b = -b \wedge C \wedge a \quad \forall a, b \in \mathbb{R}^n , \quad \forall C \in \mathcal{C}(\mathbb{R}^n) , \quad (18)$$

$$a_1 \wedge \dots \wedge a_n = \det[a_1 \dots a_n] e_{1\dots n} \quad \forall a_i \in \mathbb{R}^n . \quad (19)$$

In the last identity the expression $[a_1 \dots a_n]$ denotes a square matrix of order n , formed by the components of the n vectors arranged as columns (or rows).

These identities can be generalized, for example, the identity (16) is a particular case of the following identity

$$a \cdot B = \sum_{i=1}^k (-1)^{i-1} (a \cdot b_i) B_i \quad \forall a, b_1, \dots, b_k \in \mathbb{R}^n \quad (20)$$

where $B = b_1 \wedge \dots \wedge b_k$ and B_i is obtained from B by omitting the factor b_i . Obviously, by virtue of anticommutativity, we have $B = (-1)^{i-1} b_i \wedge B_i$.

Another important operation on a multivector A , denoted by \tilde{A} , is called the reversion. To define it, express a multivector A in the form (12) and reverse the order of the factors in the basis multivectors. This procedure gives the expression

$$\tilde{A} = \sum_{k=0}^n (-1)^{\frac{k(k-1)}{2}} \langle A \rangle_k . \quad (21)$$

In particular, the reversion has no effect on scalar or vector components.

In order to represent a k -dimensional subspace of \mathbb{R}^n and associate algebraic operations of projection and rejection, it is convenient to define a k -blade B such that the subspace \mathbb{R}^k becomes the kernel of the map

$$\mathbb{R}^n \ni a \mapsto a \wedge B \in \mathcal{C}(\mathbb{R}^n) . \quad (22)$$

By definition, a k -blade B is the product of k orthonormal vectors, namely

$$B = b_1 \dots b_k . \quad (23)$$

We always assume that $k \geq 1$. It is clear that the blade B has a geometric inverse B^{-1} given by the reversion \tilde{B} , which can be also expressed as

$$B^{-1} = b_k \dots b_1. \quad (24)$$

Indeed, $B B^{-1} = B^{-1} B = 1$ due to the orthonormality of the blade factors. Now, if the vectors $\{b_1, \dots, b_k\}$ span a subspace $\mathbb{R}^k \subset \mathbb{R}^n$, then we immediately calculate the projected (parallel) and rejected (perpendicular) components of a vector a by the following formulae

$$a_{\parallel} = (a \cdot B) B^{-1}, \quad a_{\perp} = (a \wedge B) B^{-1}. \quad (25)$$

It is straightforward to verify that $a = a_{\parallel} + a_{\perp}$ and $a_{\parallel} \cdot a_{\perp} = 0$. These simple formulae have no analogue in *Vector Algebra* for $k > 1$. In particular, for a vector a and a blade B , the identities $a B = a \cdot B + a \wedge B$ and $B a = B \cdot a + B \wedge a$ take place, which generalize the identity $a b = a \cdot b + a \wedge b$ directly following from the definitions (4).

Since the dimensions of $\mathcal{C}_k(\mathbb{R}^n)$ and $\mathcal{C}_{n-k}(\mathbb{R}^n)$ are equal, there exists a natural isomorphism called the dual operator, which is closely related to the Hodge star operator. The dual operator is defined by the following map of basis k -vectors to basis $(n-k)$ -vectors

$$*e_{i_1 i_2 \dots i_k} = (-1)^{|\sigma|} e_{j_1 j_2 \dots j_{n-k}} \quad (26)$$

where the compound index $J = \{j_1, j_2, \dots, j_{n-k}\}$ is uniquely determined by the condition $J = \{1, 2, \dots, n\} \setminus I$ with $I = \{i_1, i_2, \dots, i_k\}$, and $|\sigma|$ is the parity of the permutation

$$\sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ i_1 & i_2 & \dots & i_k & j_1 & j_2 & \dots & j_{n-k} \end{pmatrix}. \quad (27)$$

Recall that $1 \leq i_1 < i_2 < \dots < i_n \leq n$ and $1 \leq j_1 < j_2 < \dots < j_{n-k} \leq n$. In particular, we immediately obtain that $*1 = e_{12\dots n}$ and $*e_{12\dots n} = 1$. It is straightforward to verify that the inverse dual operator on basis multivectors has the expression

$$*^{-1}e_{i_1 i_2 \dots i_k} = (-1)^{|\tau|} e_{j_1 j_2 \dots j_{n-k}} \quad (28)$$

where $|\tau|$ is the parity of the permutation

$$\tau = \begin{pmatrix} 1 & 2 & \dots & n \\ j_1 & j_2 & \dots & j_{n-k} & i_1 & i_2 & \dots & i_k \end{pmatrix}. \quad (29)$$

The dual operator (and its inverse) is extended from the basis multivectors to the entire Clifford algebra by linearity.

Note that if A is a k -blade, $1 \leq k < n$, then $B = *A$ is a $(n-k)$ -blade representing the orthogonal complement of the k -dimensional vector subspace defined by A . In particular, the product AB is a nonzero pseudoscalar or, equivalently, a n -blade.

Let us define the anti-wedge product \vee in terms of the wedge product \wedge and the dual operator by the formula

$$A \vee B = *^{-1}(*A \wedge *B). \quad (30)$$

Since the product \wedge increases and \vee decreases grades, the wedge and anti-wedge products are also called the progressive and regressive outer products, respectively. It is important to stress that the regressive outer product is also associative. Indeed, since

$$(A \vee B) \vee C = *^{-1}(*A \wedge *B \wedge *C) = A \vee (B \vee C) , \quad (31)$$

the associativity of the regressive outer product is the direct consequence of the associativity of the progressive outer product.

If we keep only the progressive outer product \wedge in a Clifford algebra but all the other products are forgotten, the resulting algebraic structure is called a Grassmann algebra. The Grassmann algebra is associative, and can be constructed with no explicit use of a scalar product in the original vector space. For example, the outer product can be defined in terms of an alternating tensor product. Note that the dual isomorphism in a Clifford algebra relies on the existence of a non-degenerate scalar product (not necessarily positive definite). In Grassmann algebra, which has less structure because the scalar product may not naturally exist, a similar duality can be established by specifying a nonzero pseudoscalar, a volume element, which particularly defines an orientation of the vector space. In essence, this leads to the definition of the Grassmann–Cayley algebra which is equipped with the progressive and regressive outer products.

2.1 Clifford algebra of 3-dimensional vector space

Consider a 3-dimensional vector space \mathbb{R}^3 and generate the corresponding Clifford algebra $\mathcal{C}(\mathbb{R}^3)$ spanned by 8 basis multivectors. The multiplication table of the basis multivectors is presented in Table A1, and the progressive outer products are given in Table A2.

The dual operator is calculated on the basis multivectors in a straightforward manner, namely

$$*1 = e_{123} , \quad *e_{123} = 1 , \quad (32)$$

$$*e_1 = e_{23} , \quad *e_2 = -e_{13} , \quad *e_3 = e_{12} , \quad (33)$$

$$*e_{12} = e_3 , \quad *e_{13} = -e_2 , \quad *e_{23} = e_1 . \quad (34)$$

In particular, we obtain the expression: $*^{-1} = *$.

For any two 1-vectors a, b we get the identity $*(a \wedge b) = a \times b$ where the symbol \times denotes the cross product in a 3-dimensional Euclidean space. Note that the cross product is not associative, whereas the outer product is. The dual operator allows one to calculate the regressive outer products of the basis multivectors. The result is shown in Table A3.

It is seen that the scalar unit 1 and pseudoscalar unit e_{123} commute with all the other basis multivectors with respect to the geometric product, and therefore the subspace spanned by $\{1, e_{123}\}$ forms the central subalgebra of the Clifford algebra $\mathcal{C}(\mathbb{R}^3)$, which is isomorphic to the set of complex numbers \mathbb{C} . Actually, this central subalgebra is the centre of the algebra. The pseudoscalar unit e_{123} plays the rôle of the imaginary unit. In general, any element $a \in \mathcal{C}(\mathbb{R}^3)$ squaring to -1 (for example $a = e_{12}$) generates a 2-dimensional central subalgebra spanned by $\{1, a\}$, which is isomorphic to \mathbb{C} . However, such a subalgebra is not central.

The even subalgebra $\mathcal{C}^+(\mathbb{R}^3)$ is isomorphic to the algebra of quaternions \mathbb{H} spanned by the basis $\{1, \mathbf{i}, \mathbf{j}, \mathbf{k}\}$ satisfying the identities

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1, \quad \mathbf{i}\mathbf{j} = \mathbf{k} = -\mathbf{j}\mathbf{i}, \quad \mathbf{j}\mathbf{k} = \mathbf{i} = -\mathbf{k}\mathbf{j}, \quad \mathbf{k}\mathbf{i} = \mathbf{j} = -\mathbf{i}\mathbf{k}. \quad (35)$$

The isomorphism is established by the correspondence

$$\mathbf{i} = -e_{23} \equiv - * e_1, \quad \mathbf{j} = e_{13} \equiv - * e_2, \quad \mathbf{k} = -e_{12} \equiv - * e_3. \quad (36)$$

The inverse of a nonzero quaternion $q \in \mathcal{C}^+(\mathbb{R}^3)$ is given by the formula

$$q^{-1} = \frac{\tilde{q}}{q\tilde{q}} \quad (37)$$

where the product $q\tilde{q}$ is obviously a positive scalar. The quaternion q induces a rotation of the vector space \mathbb{R}^3 by the map $a \mapsto \rho_q(a) = q a q^{-1}$. Indeed, it is straightforward to demonstrate that $\rho_q(a) \in \mathbb{R}^3$ and $\rho_q(a) \cdot \rho_q(b) = a \cdot b$ for all $a, b \in \mathbb{R}^3$. In addition, this map preserves the space orientation: $\rho_q(c) = \rho_q(a) \times \rho_q(b)$ whenever $c = a \times b$.

Note that the quaternion q is determined by the rotation ρ_q up to a nonzero scale. In particular, the multiplicative group of normalized quaternions, which is a unit sphere $\{q\tilde{q} = 1\}$ in the four-dimensional space of quaternion components, is a double covering of the group of rotations of a three-dimensional Euclidean space ($\rho_q = \rho_{-q}$). The map $q \mapsto \rho_q$ of the group of normalized quaternions to the group of rotations is a group homomorphism since $\rho_{q_1 q_2} = \rho_{q_1} \rho_{q_2}$ [Lang 2002].

There are several other subalgebras such as that spanned by $\{1, e_1, e_2, e_{12}\}$, which is isomorphic to $\mathcal{C}(\mathbb{R}^2)$. In general, any pair of orthonormal vectors generates a subalgebra of $\mathcal{C}(\mathbb{R}^3)$, which is isomorphic to $\mathcal{C}(\mathbb{R}^2)$.

The less trivial example is the commutative subalgebra spanned by $\{1, e_1, e_{23}, e_{123}\}$, which corresponds to tessarines also called bicomplex numbers. In general, any vector a squaring to 1 generates a tessarine subalgebra spanned by $\{1, a, a e_{123}, e_{123}\}$.

2.2 Clifford algebra of 4-dimensional vector space

Consider a 4-dimensional vector space \mathbb{R}^4 and generate the corresponding Clifford algebra $\mathcal{C}(\mathbb{R}^4)$ spanned by 16 basis multivectors. The multiplication table of the basis multivectors is presented in Table A4, and the progressive outer products are given in Table A5. The dual operator is calculated on the basis multivectors as follows

$$*1 = e_{1234}, \quad *e_{1234} = 1, \quad (38)$$

$$*e_1 = e_{234}, \quad *e_2 = -e_{134}, \quad *e_3 = e_{124}, \quad *e_4 = -e_{123}, \quad (39)$$

$$*e_{123} = e_4, \quad *e_{124} = -e_3, \quad *e_{134} = e_2, \quad *e_{234} = -e_1, \quad (40)$$

$$*e_{12} = e_{34}, \quad *e_{13} = -e_{24}, \quad *e_{14} = e_{23}, \quad (41)$$

$$*e_{23} = e_{14}, \quad *e_{24} = -e_{13}, \quad *e_{34} = e_{12}. \quad (42)$$

Note that $*^{-1} \neq *$. The 1-vectors are dual to 3-vectors, whereas the 2-vectors are self-dual. There are many subalgebras of $\mathcal{C}(\mathbb{R}^4)$, such as $\mathcal{C}(\mathbb{R}^3)$ generated by any three orthonormal vectors of \mathbb{R}^4 . In particular, Table A1 is a subtable of Table A4.

This algebra naturally arises in *Projective Geometry* that models the 3-dimensional Euclidean space completed with a plane at infinity using four homogeneous coordinates. Note that in *Physics* a 4-dimensional vector space equipped with an indefinite scalar product with signature $(+, -, -, -)$ models Minkowski spacetime, so the multiplication tables and the dual operator are computed differently. Alternatively, the Minkowski spacetime can be identified with the subspace $\mathcal{C}_0(\mathbb{R}^3) \oplus \mathcal{C}_1(\mathbb{R}^3)$ whose elements are called paravectors.

3 Projective geometry

Consider an n -dimensional projective space \mathbb{P}^n over real numbers \mathbb{R} , which is identified with the set of all straight lines through the origin in \mathbb{R}^{n+1} [Casse 2006]. It is convenient to introduce homogeneous coordinates $(x_1, x_2, \dots, x_{n+1})$ specifying a particular line in \mathbb{R}^{n+1} , a point of \mathbb{P}^n , assuming that at least one of the x_i is nonzero. A point of \mathbb{P}^n is defined by $(x_1, x_2, \dots, x_{n+1})$ up to a nonzero scale. Even if we normalize the homogeneous coordinates by the condition

$$x_1^2 + x_2^2 + \dots + x_{n+1}^2 = 1, \quad (43)$$

the corresponding line will still be defined by $(x_1, x_2, \dots, x_{n+1})$ up to a factor ± 1 . So, from the topological point of view, a projective space \mathbb{P}^n is obtained from an n -dimensional sphere defined by (43) by identifying (glueing) diametrically opposite points. For a point $X \in \mathbb{P}^n$ we denote homogeneous coordinates by $\hat{X} = (x_1, x_2, \dots, x_{n+1}) \in \mathbb{R}^{n+1}$ keeping in mind the unimportance of an overall scaling.

It is worthwhile noting that the projective space \mathbb{P}^n includes \mathbb{R}^n through the following map called homogenization

$$\mathbb{R}^n \ni (x_1, x_2, \dots, x_n) \mapsto (x_1, x_2, \dots, x_n, 1) \in \mathbb{R}^{n+1}. \quad (44)$$

The inverse projection called dehomogenization is defined by the map

$$\mathbb{R}^{n+1} \ni (x_1, x_2, \dots, x_{n+1}) \mapsto \left(\frac{x_1}{x_{n+1}}, \frac{x_2}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right) \in \mathbb{R}^n \quad (45)$$

which makes sense only if $x_{n+1} \neq 0$. By definition, a point of \mathbb{P}^n lies at infinity if $x_{n+1} = 0$.

Construct the Clifford algebra $\mathcal{C}(\mathbb{R}^{n+1})$ using the set of homogeneous coordinates as the underlying vector space. Note that two points X_1, X_2 of \mathbb{P}^n are equal if and only if $\hat{X}_1 \wedge \hat{X}_2 = 0$. Let X_1, X_2 be two distinct points of \mathbb{P}^n represented by homogeneous coordinates \hat{X}_1, \hat{X}_2 . In particular, the 1-vectors \hat{X}_1, \hat{X}_2 are not collinear. Define the nonzero 2-vector $\hat{L} = \hat{X}_1 \wedge \hat{X}_2$ which represents a two-dimensional plane in \mathbb{R}^{n+1} passing through \hat{X}_1, \hat{X}_2 and the origin. The dehomogenization (45) maps this plane to a line L in \mathbb{R}^n passing through X_1 and X_2 . If both X_1 and X_2 are at infinity, the line L will be also at infinity, therefore L is a line in \mathbb{P}^n . This line as a set is defined by the kernel of the map

$$\mathbb{R}^{n+1} \ni \hat{X} \mapsto \hat{X} \wedge \hat{L} \in \mathcal{C}_3(\mathbb{R}^{n+1}). \quad (46)$$

Clearly, $\hat{X}_1 \wedge \hat{L} = \hat{X}_2 \wedge \hat{L} = 0$ or, equivalently, $X_1, X_2 \in L$ by construction. Note that the overall scales of the homogeneous coordinates do not change the line as a set.

Likewise, three points X_1, X_2, X_3 of a projective space \mathbb{P}^n whose vectors of homogeneous coordinates are not coplanar with the origin of \mathbb{R}^{n+1} define a two-dimensional plane P in \mathbb{P}^n by the nonzero 3-vector $\hat{P} = \hat{X}_1 \wedge \hat{X}_2 \wedge \hat{X}_3$ (we assume $n > 2$). This plane as a set is defined by the kernel of the map

$$\mathbb{R}^{n+1} \ni \hat{X} \mapsto \hat{X} \wedge \hat{P} \in \mathcal{C}_4(\mathbb{R}^{n+1}) . \quad (47)$$

Clearly, $\hat{X}_1 \wedge \hat{P} = \hat{X}_2 \wedge \hat{P} = \hat{X}_3 \wedge \hat{P} = 0$, thus reinstating the fact that $X_1, X_2, X_3 \in P$ by construction. A plane P can be also obtained from a line L represented by a 2-vector \hat{L} and a point X represented by a 1-vector \hat{X} using the product $\hat{P} = \hat{L} \wedge \hat{X}$.

In other words, the progressive outer product provides an efficient way to build objects of higher dimension from objects of lower dimension. This operation is called a ‘join’. The complementary operation called a ‘meet’ is handled by the regressive outer product, and corresponds to the intersection of objects. It is worthwhile emphasizing that, though the algebraic operations of Clifford algebra are always carried out, the result can be zero. This situation indicates that the resulting object is invalid. For example, this happens when two identical objects are multiplied.

3.1 Incidence relations in projective plane \mathbb{P}^2

Consider two distinct lines L_1 and L_2 in \mathbb{P}^2 represented by two 2-vectors in $\mathcal{C}(\mathbb{R}^3)$. Compute the regressive product $\hat{X} = \hat{L}_1 \vee \hat{L}_2$ which is obviously a 1-vector and therefore represents a point X of \mathbb{P}^2 . It is straightforward to prove that X is the intersection point of the lines L_1 and L_2 possibly lying at infinity. Also note that a point X belongs to a line L if either $\hat{X} \wedge \hat{L} = 0$ or $\hat{X} \vee \hat{L} = 0$.

Two points X_1, X_2 of the projective plane \mathbb{P}^2 are equal if either $\hat{X}_1 \wedge \hat{X}_2 = 0$ or $\hat{X}_1 \vee \hat{X}_2 = 0$. Likewise, two lines L_1, L_2 are equal if either $\hat{L}_1 \wedge \hat{L}_2 = 0$ or $\hat{L}_1 \vee \hat{L}_2 = 0$. There is a natural duality between points and lines established by the mutual correspondences $\hat{L} = *\hat{X}$ and $\hat{X} = *\hat{L}$.

Let A_1, B_1, C_1 and A_2, B_2, C_2 be two triplets of collinear points in the projective plane \mathbb{P}^2 . Algebraically, this means that $\hat{A}_1 \wedge \hat{B}_1 \wedge \hat{C}_1 = 0$ and $\hat{A}_2 \wedge \hat{B}_2 \wedge \hat{C}_2 = 0$. Then the intersection points A, B, C constructed by the rules

$$\hat{A} = (\hat{B}_1 \wedge \hat{C}_2) \vee (\hat{B}_2 \wedge \hat{C}_1) , \quad (48)$$

$$\hat{B} = (\hat{C}_1 \wedge \hat{A}_2) \vee (\hat{C}_2 \wedge \hat{A}_1) , \quad (49)$$

$$\hat{C} = (\hat{A}_1 \wedge \hat{B}_2) \vee (\hat{A}_2 \wedge \hat{B}_1) , \quad (50)$$

must be also collinear, that is $\hat{A} \wedge \hat{B} \wedge \hat{C} = 0$. This is the statement of the theorem of *Pappus*. A dual version of this theorem exists in which collinear points are replaced with concurrent lines.

Now assume that the points A_1, B_1, C_1 and A_2, B_2, C_2 are not collinear but form two non-degenerate triangles. Construct the intersection points A, B, C of the corresponding triangle sides by the rules

$$\hat{A} = (\hat{B}_1 \wedge \hat{C}_1) \vee (\hat{B}_2 \wedge \hat{C}_2), \quad (51)$$

$$\hat{B} = (\hat{C}_1 \wedge \hat{A}_1) \vee (\hat{C}_2 \wedge \hat{A}_2), \quad (52)$$

$$\hat{C} = (\hat{A}_1 \wedge \hat{B}_1) \vee (\hat{A}_2 \wedge \hat{B}_2), \quad (53)$$

and compute the pseudoscalar $p = \hat{A} \wedge \hat{B} \wedge \hat{C}$ which vanishes if and only if the intersection points are collinear. Draw three lines L_a, L_b, L_c through the corresponding vertices of the triangles, namely

$$\hat{L}_a = \hat{A}_1 \wedge \hat{A}_2, \quad \hat{L}_b = \hat{B}_1 \wedge \hat{B}_2, \quad \hat{L}_c = \hat{C}_1 \wedge \hat{C}_2, \quad (54)$$

and compute the scalar $q = \hat{L}_a \vee \hat{L}_b \vee \hat{L}_c$ which vanishes if and only if the lines are concurrent. The theorem of *Desargues* states that p and q vanish simultaneously. This statement follows from the explicit formula

$$*p = -I_1 I_2 q, \quad I_1 = \hat{A}_1 \wedge \hat{B}_1 \wedge \hat{C}_1, \quad I_2 = \hat{A}_2 \wedge \hat{B}_2 \wedge \hat{C}_2, \quad (55)$$

in combination with the fact that the pseudoscalars I_1, I_2 do not vanish (recall that the triangles are not degenerate).

3.2 Incidence relations in projective space \mathbb{P}^3

Now consider a line L and a plane P in \mathbb{P}^3 represented respectively by an appropriate 2-vector \hat{L} and a 3-vector \hat{P} in $\mathcal{C}(\mathbb{R}^4)$. Then the product $\hat{X} = \hat{L} \vee \hat{P}$, which is obviously a 1-vector, represents the intersection point X . The intersection of two planes, P_1 and P_2 , is given by the product $\hat{L} = \hat{P}_1 \vee \hat{P}_2$ which is a 2-vector in $\mathcal{C}(\mathbb{R}^4)$ and therefore represents a line L . The point of intersection of three planes is represented by the triple product $\hat{X} = \hat{P}_1 \vee \hat{P}_2 \vee \hat{P}_3$ which is obviously a 1-vector. These operations work for general (non-degenerate) configurations.

Using either the regressive or progressive product, multiply a point and a plane to get a signed minimum distance between them. Likewise, multiply two lines L_1 and L_2 to get a special signed crossing value. Depending on the product, the result will be either a scalar or pseudoscalar. If the crossing value vanishes, say $\hat{L}_1 \wedge \hat{L}_2 = 0$, the lines are coplanar and hence intersect. The homogeneous coordinates of the intersection point X belong to the kernel of the map

$$\mathbb{R}^4 \ni \hat{X} \mapsto (\hat{X} \wedge \hat{L}_1, \hat{X} \wedge \hat{L}_2) \in \mathcal{C}_3(\mathbb{R}^4) \times \mathcal{C}_3(\mathbb{R}^4). \quad (56)$$

In other words, we have to find a nonzero solution to the system of equations

$$\begin{cases} \hat{X} \wedge \hat{L}_1 &= 0 \\ \hat{X} \wedge \hat{L}_2 &= 0 \end{cases} \quad (57)$$

which is equivalent to an overdetermined system of 8 homogeneous linear equations for 4 components of \hat{X} . The solution can be efficiently obtained by the Singular Value Decomposition (SVD) algorithm of *Linear Algebra* [Golub & Van Loan 1996]. The matrix equation is derived in a straightforward manner by observing that the equation $\hat{X} \wedge \hat{L} = 0$ is equivalent to

$$\begin{bmatrix} l_{23} & -l_{13} & l_{12} & 0 \\ l_{24} & -l_{14} & 0 & l_{12} \\ l_{34} & 0 & -l_{14} & l_{13} \\ 0 & l_{34} & -l_{24} & l_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (58)$$

where

$$\hat{X} = x_1 e_1 + x_2 e_2 + x_3 e_3 + x_4 e_4, \quad (59)$$

$$\hat{L} = l_{12} e_{12} + l_{13} e_{13} + l_{14} e_{14} + l_{23} e_{23} + l_{24} e_{24} + l_{34} e_{34}. \quad (60)$$

Note that the six coefficients in the line representation (60) are the Plücker coordinates.

The Plücker coordinates providing an economic parametrization of all lines in \mathbb{P}^3 are not arbitrary but satisfy a constraint imposed by the Klein quadric

$$l_{12} l_{34} - l_{13} l_{24} + l_{14} l_{23} = 0. \quad (61)$$

A line $\hat{L} = \hat{X} \wedge \hat{Y}$ constructed by two points has the expression

$$\begin{aligned} \hat{L} = & (x_1 y_2 - x_2 y_1) e_{12} + (x_1 y_3 - x_3 y_1) e_{13} + (x_1 y_4 - x_4 y_1) e_{14} \\ & + (x_2 y_3 - x_3 y_2) e_{23} + (x_2 y_4 - x_4 y_2) e_{24} + (x_3 y_4 - x_4 y_3) e_{34} \end{aligned} \quad (62)$$

which no longer contains any information about the 1-vectors \hat{X}, \hat{Y} used to create it. This is contrary to a parametric representation of a line L passing through points X, Y . It is straightforward to check that the constraint (61) is imposed for the line (62).

Two points X_1, X_2 of the projective space \mathbb{P}^3 are equal if $\hat{X}_1 \wedge \hat{X}_2 = 0$. Likewise, two planes P_1, P_2 are equal if $\hat{P}_1 \vee \hat{P}_2 = 0$. There is a natural duality between points and planes established by the correspondences $\hat{P} = *\hat{X}$ and $\hat{X} = *\hat{P}$.

The incidence relations with lines are more involved. Two lines L_1, L_2 are equal if the kernel of the map (56) is two-dimensional. This question can be efficiently answered using the SVD algorithm, or by the direct calculation of the rank r of the matrix

$$M(\hat{L}_1, \hat{L}_2) = \begin{bmatrix} l_{23}^{(1)} & -l_{13}^{(1)} & l_{12}^{(1)} & 0 \\ l_{24}^{(1)} & -l_{14}^{(1)} & 0 & l_{12}^{(1)} \\ l_{34}^{(1)} & 0 & -l_{14}^{(1)} & l_{13}^{(1)} \\ 0 & l_{34}^{(1)} & -l_{24}^{(1)} & l_{23}^{(1)} \\ l_{23}^{(2)} & -l_{13}^{(2)} & l_{12}^{(2)} & 0 \\ l_{24}^{(2)} & -l_{14}^{(2)} & 0 & l_{12}^{(2)} \\ l_{34}^{(2)} & 0 & -l_{14}^{(2)} & l_{13}^{(2)} \\ 0 & l_{34}^{(2)} & -l_{24}^{(2)} & l_{23}^{(2)} \end{bmatrix} \quad (63)$$

where $l_{ij}^{(m)}$ are the Plücker coordinates of the line \hat{L}_m . The lines are not coplanar if $r = 4$, the lines have a unique point of intersection if $r = 3$, and the lines coincide if $r = 2$. The case $r < 2$ never happens because a valid line as a set is a one-dimensional object.

4 Application to 3D reconstruction

In this section we provide a theoretical background for 3D reconstruction using *Geometric Algebra*. We derive several important concepts and relations from [Hartley & Zisserman 2003] in a simple and elegant form.

The three-dimensional space will be modelled by the projective space \mathbb{P}^3 . To apply efficient algebraic methods when building geometric objects, we generate the Clifford algebra $\mathcal{C}(\mathbb{R}^4)$. We will slightly abuse notation by using the same symbol for an object in \mathbb{P}^3 and its representation in $\mathcal{C}(\mathbb{R}^4)$. Recall that the object representation in $\mathcal{C}(\mathbb{R}^4)$ is defined up to a nonzero scale.

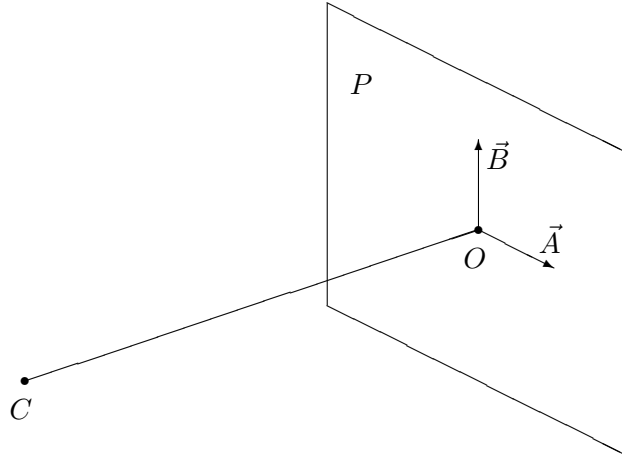


Figure 1: Pinhole camera

A pinhole camera is completely described by its optical centre C and projection plane P both considered to be objects of the projective space \mathbb{P}^3 . It is always guaranteed that C does not belong to P , which is equivalent to $C \wedge P \neq 0$. The projective space \mathbb{P}^3 has no natural metric structure. However, image points are measured in Cartesian coordinates of the projection plane which does have a metric. So, it is important to establish a parametrization of image points in \mathbb{P}^3 in terms of the Cartesian coordinates of the projection plane. This can be done, for example, by identifying some point O , say the principal point, with the origin of the coordinate system and choosing two points A, B on the axes of the orthogonal coordinates. The four points C, O, A, B completely determine the pinhole camera, in particular $P = O \wedge A \wedge B$. A point X in the projection plane P has the representation $X = aA + bB + cO$ where the scalars a, b, c are defined up to a common scale. These scalars are directly related to the Cartesian coordinates. Indeed, since all the points X, O, A, B are finite (not lying at infinity), we can dehomogenize them and set $a + b + c = 1$ without loss of generality. As a result, we get the representation $X = O + a(A - O) + b(B - O)$ where a, b are the Cartesian coordinates. The direction vectors $\vec{A} = A - O$ and $\vec{B} = B - O$ belong to the plane at infinity of \mathbb{P}^3 , and are orthogonal by construction. The Cartesian coordinates are now readily computed by the formulae

$$a = \frac{\vec{A} \cdot \vec{X}}{\vec{A} \cdot \vec{A}}, \quad b = \frac{\vec{B} \cdot \vec{X}}{\vec{B} \cdot \vec{B}}, \quad (64)$$

where $\vec{X} = X - O$ is the direction vector. The scalar product is induced by the projection plane P .

It is important to emphasize that it makes sense to talk about orthogonality of particular vectors constructed from the projection plane P . In fact, most problems of 3D reconstruction are formulated as a mathematical problem of fitting a projective (or affine) variety [Cox, Little & O'Shea 2007] to measurement points by minimizing a cost function associated with the metric structure of projection planes.

Let $X \in \mathbb{P}^3$ be an arbitrary point different from the optical centre C . The following map projects the point X to the projection plane P

$$X \mapsto P \vee (C \wedge X) . \quad (65)$$

This map generates the camera matrix when the projected point is represented in the Cartesian coordinates of the projection plane P followed by the homogenization procedure. The size of the camera matrix is 3-by-4 as it maps four homogeneous coordinates of the point X to the three homogeneous coordinates a, b, c of the projection plane.

Now let L be an arbitrary line in \mathbb{P}^3 not containing the optical centre C . The projection of this line to the plane P is given by the similar map

$$L \mapsto P \vee (C \wedge L) . \quad (66)$$

The result is the unique line of intersection of two planes.

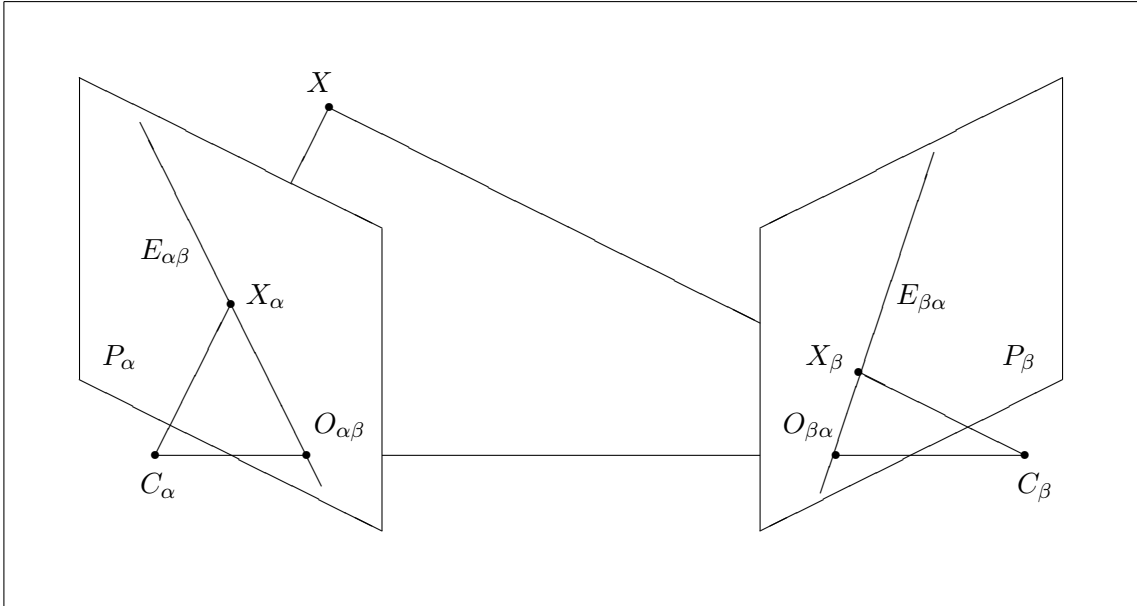


Figure 2: Epipolar geometry

In multiview geometry we are provided with a sequence of optical centres and projection planes which we label by some index. Given two camera configurations (C_α, P_α) and (C_β, P_β) we construct the base line $C_\alpha \wedge C_\beta$ and intersect it with the projection planes (see Figure 2). The intersection points are respectively the epipoles $O_{\alpha\beta}$ and $O_{\beta\alpha}$. The epipolar

lines $E_{\alpha\beta}$ and $E_{\beta\alpha}$ are formed by the intersection of a plane containing the baseline with the projection planes. Obviously, all the epipolar lines in a projection plane are concurrent as they contain the epipole, and therefore have the geometry of the projective line \mathbb{P}^1 .

There exists the fundamental mapping $\Phi_{\alpha\beta}$ between the epipolar lines, induced by the pencil of planes containing the baseline. Algebraically, the fundamental mapping is given by the formula

$$\Phi_{\alpha\beta} : \mathbb{P}^1 \ni E_{\alpha\beta} \mapsto E_{\beta\alpha} = (C_\alpha \wedge E_{\alpha\beta}) \vee P_\beta \in \mathbb{P}^1. \quad (67)$$

It is straightforward to prove that the mapping $\Phi_{\alpha\beta}$ is well defined (recall that $C_\beta \notin P_\beta$) and is invertible. Indeed, by virtue of symmetry, we have $\Phi_{\beta\alpha} = \Phi_{\alpha\beta}^{-1}$.

It is clear that, if $X_\alpha \in P_\alpha$ and $X_\beta \in P_\beta$ are two images of some point X , then $X_\alpha \in E_{\alpha\beta}$ and $X_\beta \in E_{\beta\alpha}$ as shown in Figure 2. Algebraically, this is equivalent to the constraint

$$\Phi_{\alpha\beta}(X_\alpha \wedge O_{\alpha\beta}) \wedge X_\beta = 0 \quad (68)$$

where the epipole has the expression $O_{\alpha\beta} = P_\alpha \vee (C_\alpha \wedge C_\beta)$. This equation produces the fundamental matrix [Hartley & Zisserman 2003] when X_α and X_β are respectively represented in coordinates of the planes P_α and P_β .

Given two image points X_α, X_β satisfying the constraint (68), the point X is recovered by the intersection of the optical rays $C_\alpha \wedge X_\alpha$ and $C_\beta \wedge X_\beta$. This operation called triangulation is well defined if the optical rays are coplanar. The later is equivalent to (68) but a simpler form which does not involve the projection plane P_β is as follows

$$X_\alpha \wedge C_\alpha \wedge C_\beta \wedge X_\beta = 0. \quad (69)$$

The triangulation problem reduces to the system of equations

$$\begin{cases} X \wedge C_\alpha \wedge X_\alpha &= 0 \\ X \wedge C_\beta \wedge X_\beta &= 0 \end{cases} \quad (70)$$

which is of the form (57). A nonzero solution (up to a scaling factor) exists provided that the constraint (69) is imposed. The conditions for $X_\alpha \in P_\alpha$ and $X_\beta \in P_\beta$ are equivalent to the following

$$X_\alpha \wedge P_\alpha = 0, \quad X_\beta \wedge P_\beta = 0. \quad (71)$$

Here the progressive product in the inclusion tests can be replaced with the regressive product, since points and planes are in the duality relationship.

It is worthwhile emphasizing that the projection planes do not appear explicitly in the triangulation problem. This is because we consider the image points belonging to the projective space \mathbb{P}^3 .

The triangulation procedure solves the following problem of 3D reconstruction in a closed form: “Given two camera configurations and arrays of pairs of matching image points satisfying the constraint (69) exactly, find the corresponding spatial points.”

However, in reality, the 3D reconstruction procedure is complicated by the presence of unavoidable noise in the measurements of image points. In particular, we cannot assume

any longer that the constraint (69) is exactly satisfied. Another problem is associated with *a priori* unknown camera configurations. Finally, the matching points when automatically produced may not be in correspondence at all. Such points are called outliers as opposed to inliers, and must be eliminated. The fundamental matrix plays the crucial rôle in the elimination procedure [Hartley & Zisserman 2003].

In the two-view geometry the reconstruction of lines is less restrictive. Let L_α and L_β be two image lines in the planes P_α and P_β , not containing the epipoles $O_{\alpha\beta}$ and $O_{\beta\alpha}$, respectively. In other words, neither of them is an epipolar line. Then the reconstructed line L is given by the formula

$$L = (C_\alpha \wedge L_\alpha) \vee (C_\beta \wedge L_\beta) . \quad (72)$$

The result is the unique line of intersection of two different planes. However, this advantage turns out to be disadvantage as there is no way to eliminate outlying lines. The situation becomes different in the three-view geometry [Hartley & Zisserman 2003].

If both projection lines are epipolar lines, then the planes are either equal or intersect at the base line. The base line is obviously projected to the epipoles of the projection planes, hence this situation cannot generate image lines. If only one projection line is epipolar, say L_α , but the other line L_β is not, then both planes $C_\alpha \wedge L_\alpha$ and $C_\beta \wedge L_\beta$ contain the optical centre C_β . Therefore, the reconstructed line L contains C_β and must be projected to the epipole $O_{\beta\alpha}$. This can be reformulated in the following way. If a line L contains the optical centre of one view, then in the other view it must be projected to either an epipolar line or the epipole.

As in the triangulation problem, the projection planes do not appear explicitly in the solution. The conditions for $L_\alpha \subset P_\alpha$ and $L_\beta \subset P_\beta$ are equivalent to the following

$$L_\alpha \vee P_\alpha = 0 , \quad L_\beta \vee P_\beta = 0 . \quad (73)$$

Here the regressive product in the inclusion tests cannot be replaced with the progressive product.

5 Description of the developed MATLAB code

In this section we briefly describe the MATLAB code developed. Basic algebraic operations of the Clifford algebra, currently supporting dimensions up to $n = 4$, are implemented in the `clifford_algebra.m` file whose content is provided in Appendix B. This reusable MATLAB function has to be placed on the MATLAB path to be called from different directories.

For the sake of code performance, we have deliberately avoided the construction of MATLAB classes. Instead, a collection of multivectors is represented by an ordinary matrix containing 2^n columns. The fully vectorized code is capable of operating on large arrays of multivectors in an efficient way. Factors in binary products can be either arrays of multivectors of the same number of rows or an array and a single multivector represented by a row matrix.

The unit tests of the Clifford algebra implementation are provided in Appendix C. The basic properties of algebraic operations and known identities derived in Clifford algebra are thoroughly tested using randomly generated arrays of multivectors. For example, we have validated the associativity of geometric, progressive and regressive products, the basic properties of scalar, inner and cross products, and the properties of subalgebras including even subalgebra, complex and bicomplex numbers, and quaternions. The introduced operations, such as the dual, reverse and blade operations, are also tested. The last test displays the performance of the vectorized geometric product against its serial counterpart.

The application of Clifford algebra to *Projective Geometry* is given in Appendix D. The unit tests include the validation of basic incidence relations and celebrated theorems of *Projective Geometry* such as those of *Pappus* and *Desargues*. Also, epipolar geometry, fundamental map, and the point and line reconstruction procedures are tested.

6 Discussion

We have presented a fully vectorized MATLAB code for the basic operations of *Geometric Algebra* whose real power manifests itself when particularly combined with *Projective Geometry*. The developed code is thoroughly tested and can be used in various applications which involve complicated geometric constructs. The vectorized code can be applied to real-time simulations when there is a requirement to operate on large arrays of multivectors. For example, quaternions naturally embedded in the Clifford algebra $\mathcal{C}(\mathbb{R}^3)$ can be utilized to implement fast rotation of large arrays of 3D objects.

Having mainly in mind the application of the Clifford algebra to the reconstruction of a three-dimensional structure from a sequence of images, we confined ourselves to a positive definite scalar product when defining the algebra multiplication rules. The extension of the code to an indefinite scalar product which arises in the geometry of spacetime is straightforward.

Acknowledgement

The author is grateful to Dr Leszek Swierkowski from DSTO for helpful discussion and valuable comments.

References

- Casse, R. (2006) *Projective Geometry: An Introduction*, Oxford University Press, Oxford.
- Corrochano, E. B. & Sobczyk, G., eds (2001) *Geometric Algebra with Applications in Science and Engineering*, Birkhäuser, Boston.
- Cox, D., Little, J. & O'Shea, D. (2007) *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, 3rd edn, Springer, New York.
- Garling, D. J. H. (2011) *Clifford Algebras: An Introduction*, Cambridge University Press, Cambridge.
- Golub, G. H. & Van Loan, C. F. (1996) *Matrix Computations*, 3rd edn, The John Hopkins University Press, Baltimore.
- Hartley, R. & Zisserman, A. (2003) *Multiple View Geometry in Computer Vision*, 2nd edn, Cambridge University Press, Cambridge.
- Hestenes, D. & Sobczyk, G. (1987) *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*, Vol. 5 of *Fundamental Theories of Physics*, Springer, Berlin.
- Hurley, D. J. & Vandyck, M. A. (2000) *Geometry, Spinors and Applications*, Springer, Berlin.
- Lang, S. (2002) *Algebra*, Vol. 211 of *Graduate Texts in Mathematics*, revised 3rd edn, Springer, New York.
- Perwass, C. (2009) *Geometric Algebra with Applications in Engineering*, Springer, Berlin.

Appendix A Multiplication tables

The multiplication tables of two Clifford algebras, $\mathcal{C}(\mathbb{R}^3)$ and $\mathcal{C}(\mathbb{R}^4)$, are given in this appendix. The basis multivectors in the leftmost column and in the top row are respectively the left and right factors (multiplicand and multiplier) of the binary products.

Table A1: Geometric products of the basis multivectors of $\mathcal{C}(\mathbb{R}^3)$

	1	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}
1	1	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}
e_1	e_1	1	e_{12}	e_{13}	e_2	e_3	e_{123}	e_{23}
e_2	e_2	$-e_{12}$	1	e_{23}	$-e_1$	$-e_{123}$	e_3	$-e_{13}$
e_3	e_3	$-e_{13}$	$-e_{23}$	1	e_{123}	$-e_1$	$-e_2$	e_{12}
e_{12}	e_{12}	$-e_2$	e_1	e_{123}	-1	$-e_{23}$	e_{13}	$-e_3$
e_{13}	e_{13}	$-e_3$	$-e_{123}$	e_1	e_{23}	-1	$-e_{12}$	e_2
e_{23}	e_{23}	e_{123}	$-e_3$	e_2	$-e_{13}$	e_{12}	-1	$-e_1$
e_{123}	e_{123}	e_{23}	$-e_{13}$	e_{12}	$-e_3$	e_2	$-e_1$	-1

Table A2: Progressive outer products of the basis multivectors of $\mathcal{C}(\mathbb{R}^3)$

\wedge	1	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}
1	1	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}
e_1	e_1	0	e_{12}	e_{13}	0	0	e_{123}	0
e_2	e_2	$-e_{12}$	0	e_{23}	0	$-e_{123}$	0	0
e_3	e_3	$-e_{13}$	$-e_{23}$	0	e_{123}	0	0	0
e_{12}	e_{12}	0	0	e_{123}	0	0	0	0
e_{13}	e_{13}	0	$-e_{123}$	0	0	0	0	0
e_{23}	e_{23}	e_{123}	0	0	0	0	0	0
e_{123}	e_{123}	0	0	0	0	0	0	0

Table A3: Regressive outer products of the basis multivectors of $\mathcal{C}(\mathbb{R}^3)$

\vee	1	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}
1	0	0	0	0	0	0	0	1
e_1	0	0	0	0	0	0	1	e_1
e_2	0	0	0	0	0	-1	0	e_2
e_3	0	0	0	0	1	0	0	e_3
e_{12}	0	0	0	1	0	e_1	e_2	e_{12}
e_{13}	0	0	-1	0	$-e_1$	0	e_3	e_{13}
e_{23}	0	1	0	0	$-e_2$	$-e_3$	0	e_{23}
e_{123}	1	e_1	e_2	e_3	e_{12}	e_{13}	e_{23}	e_{123}

Table A4: Geometric products of the basis multivectors of $\mathcal{C}(\mathbb{R}^4)$

	1	e_1	e_2	e_3	e_4	e_{12}	e_{13}	e_{14}
1	1	e_1	e_2	e_3	e_4	e_{12}	e_{13}	e_{14}
e_1	e_1	1	e_{12}	e_{13}	e_{14}	e_2	e_3	e_4
e_2	e_2	$-e_{12}$	1	e_{23}	e_{24}	$-e_1$	$-e_{123}$	$-e_{124}$
e_3	e_3	$-e_{13}$	$-e_{23}$	1	e_{34}	e_{123}	$-e_1$	$-e_{134}$
e_4	e_4	$-e_{14}$	$-e_{24}$	$-e_{34}$	1	e_{124}	e_{134}	$-e_1$
e_{12}	e_{12}	$-e_2$	e_1	e_{123}	e_{124}	-1	$-e_{23}$	$-e_{24}$
e_{13}	e_{13}	$-e_3$	$-e_{123}$	e_1	e_{134}	e_{23}	-1	$-e_{34}$
e_{14}	e_{14}	$-e_4$	$-e_{124}$	$-e_{134}$	e_1	e_{24}	e_{34}	-1
e_{23}	e_{23}	e_{123}	$-e_3$	e_2	e_{234}	$-e_{13}$	e_{12}	e_{1234}
e_{24}	e_{24}	e_{124}	$-e_4$	$-e_{234}$	e_2	$-e_{14}$	$-e_{1234}$	e_{12}
e_{34}	e_{34}	e_{134}	e_{234}	$-e_4$	e_3	e_{1234}	$-e_{14}$	e_{13}
e_{123}	e_{123}	e_{23}	$-e_{13}$	e_{12}	e_{1234}	$-e_3$	e_2	e_{234}
e_{124}	e_{124}	e_{24}	$-e_{14}$	$-e_{1234}$	e_{12}	$-e_4$	$-e_{234}$	e_2
e_{134}	e_{134}	e_{34}	e_{1234}	$-e_{14}$	e_{13}	e_{234}	$-e_4$	e_3
e_{234}	e_{234}	$-e_{1234}$	e_{34}	$-e_{24}$	e_{23}	$-e_{134}$	e_{124}	$-e_{123}$
e_{1234}	e_{1234}	$-e_{234}$	e_{134}	$-e_{124}$	e_{123}	$-e_{34}$	e_{24}	$-e_{23}$
	e_{23}	e_{24}	e_{34}	e_{123}	e_{124}	e_{134}	e_{234}	e_{1234}
1	e_{23}	e_{24}	e_{34}	e_{123}	e_{124}	e_{134}	e_{234}	e_{1234}
e_1	e_{123}	e_{124}	e_{134}	e_{23}	e_{24}	e_{34}	e_{1234}	e_{234}
e_2	e_3	e_4	e_{234}	$-e_{13}$	$-e_{14}$	$-e_{1234}$	e_{34}	$-e_{134}$
e_3	$-e_2$	$-e_{234}$	e_4	e_{12}	e_{1234}	$-e_{14}$	$-e_{24}$	e_{124}
e_4	e_{234}	$-e_2$	$-e_3$	$-e_{1234}$	e_{12}	e_{13}	e_{23}	$-e_{123}$
e_{12}	e_{13}	e_{14}	e_{1234}	$-e_3$	$-e_4$	$-e_{234}$	e_{134}	$-e_{34}$
e_{13}	$-e_{12}$	$-e_{1234}$	e_{14}	e_2	e_{234}	$-e_4$	$-e_{124}$	e_{24}
e_{14}	e_{1234}	$-e_{12}$	$-e_{13}$	$-e_{234}$	e_2	e_3	e_{123}	$-e_{23}$
e_{23}	-1	$-e_{34}$	e_{24}	$-e_1$	$-e_{134}$	e_{124}	$-e_4$	$-e_{14}$
e_{24}	e_{34}	-1	$-e_{23}$	e_{134}	$-e_1$	$-e_{123}$	e_3	e_{13}
e_{34}	$-e_{24}$	e_{23}	-1	$-e_{124}$	e_{123}	$-e_1$	$-e_2$	$-e_{12}$
e_{123}	$-e_1$	$-e_{134}$	e_{124}	-1	$-e_{34}$	e_{24}	$-e_{14}$	$-e_4$
e_{124}	e_{134}	$-e_1$	$-e_{123}$	e_{34}	-1	$-e_{23}$	e_{13}	e_3
e_{134}	$-e_{124}$	e_{123}	$-e_1$	$-e_{24}$	e_{23}	-1	$-e_{12}$	$-e_2$
e_{234}	$-e_4$	e_3	$-e_2$	e_{14}	$-e_{13}$	e_{12}	-1	e_1
e_{1234}	$-e_{14}$	e_{13}	$-e_{12}$	e_4	$-e_3$	e_2	$-e_1$	1

Table A5: Progressive outer products of the basis multivectors of $\mathcal{C}(\mathbb{R}^4)$

\wedge	1	e_1	e_2	e_3	e_4	e_{12}	e_{13}	e_{14}
1	1	e_1	e_2	e_3	e_4	e_{12}	e_{13}	e_{14}
e_1	e_1	0	e_{12}	e_{13}	e_{14}	0	0	0
e_2	e_2	$-e_{12}$	0	e_{23}	e_{24}	0	$-e_{123}$	$-e_{124}$
e_3	e_3	$-e_{13}$	$-e_{23}$	0	e_{34}	e_{123}	0	$-e_{134}$
e_4	e_4	$-e_{14}$	$-e_{24}$	$-e_{34}$	0	e_{124}	e_{134}	0
e_{12}	e_{12}	0	0	e_{123}	e_{124}	0	0	0
e_{13}	e_{13}	0	$-e_{123}$	0	e_{134}	0	0	0
e_{14}	e_{14}	0	$-e_{124}$	$-e_{134}$	0	0	0	0
e_{23}	e_{23}	e_{123}	0	0	e_{234}	0	0	e_{1234}
e_{24}	e_{24}	e_{124}	0	$-e_{234}$	0	0	$-e_{1234}$	0
e_{34}	e_{34}	e_{134}	e_{234}	0	0	e_{1234}	0	0
e_{123}	e_{123}	0	0	0	e_{1234}	0	0	0
e_{124}	e_{124}	0	0	$-e_{1234}$	0	0	0	0
e_{134}	e_{134}	0	e_{1234}	0	0	0	0	0
e_{234}	e_{234}	$-e_{1234}$	0	0	0	0	0	0
e_{1234}	e_{1234}	0	0	0	0	0	0	0

\wedge	e_{23}	e_{24}	e_{34}	e_{123}	e_{124}	e_{134}	e_{234}	e_{1234}
1	e_{23}	e_{24}	e_{34}	e_{123}	e_{124}	e_{134}	e_{234}	e_{1234}
e_1	e_{123}	e_{124}	e_{134}	0	0	0	e_{1234}	0
e_2	0	0	e_{234}	0	0	$-e_{1234}$	0	0
e_3	0	$-e_{234}$	0	0	e_{1234}	0	0	0
e_4	e_{234}	0	0	$-e_{1234}$	0	0	0	0
e_{12}	0	0	e_{1234}	0	0	0	0	0
e_{13}	0	$-e_{1234}$	0	0	0	0	0	0
e_{14}	e_{1234}	0	0	0	0	0	0	0
e_{23}	0	0	0	0	0	0	0	0
e_{24}	0	0	0	0	0	0	0	0
e_{34}	0	0	0	0	0	0	0	0
e_{123}	0	0	0	0	0	0	0	0
e_{124}	0	0	0	0	0	0	0	0
e_{134}	0	0	0	0	0	0	0	0
e_{234}	0	0	0	0	0	0	0	0
e_{1234}	0	0	0	0	0	0	0	0

Appendix B Listing of clifford_algebra.m

```
%CA = CLIFFORD_ALGEBRA(n)
%
%Clifford algebra of an n-dimensional vector space. Multivectors are
%represented by 2^n-column matrices. All algebraic operations are
%vectorized.
%
% d = CA.dimension          - algebra dimension (d = 2^n)
% c = CA.set_scalar(s)      - create multivector from scalar
% s = CA.get_scalar(c)      - extract scalar from multivector
% c = CA.set_vector(v)      - create multivector from vector
% v = CA.get_vector(c)      - extract vector from multivector
% c = CA.even(c)            - projection to the even subalgebra
% c = CA.reverse(c)         - reversion operator on multivector
% c = CA.dual(c)            - dual operator on multivector
% c = CA.dual_inv(c)         - inverse dual operator on multivector
% c = CA.product(a,b)       - geometric product of two multivectors
% c = CA.product_p(a,b)     - progressive outer product of two multivectors
% c = CA.product_r(a,b)     - regressive outer product of two multivectors
% c = CA.product_s(a,b)     - inner (scalar) product of two multivectors
% CA.display()              - display basic information
%
%Copyright (C) 2014 Defence Science and Technology Organisation
%
%Created by Leonid K. Antanovskii
function CA = clifford_algebra(n)
switch n
    case {1,2,3,4}
    otherwise
        error('Space dimension %d not supported.',n);
end
d = 2^n;

function check_dimension(c)
    if size(c,2) ~= d
        error('Wrong multivector dimension.');
```

```
    end
end

function c = set_scalar(s)
    if size(s,2) ~= 1
        error('Wrong scalar dimension.');
```

```
    end
    c = zeros(size(s,1),d);
    c(:,1) = s;
end

function c = set_vector(v)
    if size(v,2) ~= n
        error('Wrong vector dimension.');
```

```
    end
    c = zeros(size(v,1),d);
    c(:,2:n+1) = v;
end

function s = get_scalar(c)
    check_dimension(c);
    s = c(:,1);
end

function v = get_vector(c)
    check_dimension(c);
    v = c(:,2:n+1);
end
```

```

function c = reverse(c)
    check_dimension(c);
    switch n
        case 1
        case 2
            c = [c(:,1:3),-c(:,4)];
        case 3
            c = [c(:,1:4),-c(:,5:8)];
        case 4
            c = [c(:,1:5),-c(:,6:15),c(:,16)];
    end
end

function c = even(c)
    check_dimension(c);
    switch n
        case 1
            c(:,2) = 0.0;
        case 2
            c(:,[2,3]) = 0.0;
        case 3
            c(:,[2,3,4,8]) = 0.0;
        case 4
            c(:,[2,3,4,5,12,13,14,15]) = 0.0;
    end
end

function c = dual(c)
    check_dimension(c);
    switch n
        case 1
            c = dual1d(c);
        case 2
            c = dual2d(c);
        case 3
            c = dual3d(c);
        case 4
            c = dual4d(c);
    end
end

function c = dual_inv(c)
    check_dimension(c);
    switch n
        case 1
            c = dual1d(c);
        case 2
            c = dual2d_inv(c);
        case 3
            c = dual3d(c);
        case 4
            c = dual4d_inv(c);
    end
end

function c = product(a,b)
    check_dimension(a);
    check_dimension(b);
    switch n
        case 1
            c = product1d(a,b);
        case 2
            c = product2d(a,b);
        case 3
            c = product3d(a,b);
        case 4
            c = product4d(a,b);
    end
end

```

```

    end
end

function c = product_p(a,b)
    check_dimension(a);
    check_dimension(b);
    switch n
        case 1
            c = product1d_p(a,b);
        case 2
            c = product2d_p(a,b);
        case 3
            c = product3d_p(a,b);
        case 4
            c = product4d_p(a,b);
    end
end

function c = product_r(a,b)
    check_dimension(a);
    check_dimension(b);
    switch n
        case 1
            c = product1d_r(a,b);
        case 2
            c = product2d_r(a,b);
        case 3
            c = product3d_r(a,b);
        case 4
            c = product4d_r(a,b);
    end
end

function c = product_s(a,b)
    check_dimension(a);
    check_dimension(b);
    switch n
        case 1
            c = product1d(a,b);
        case 2
            c = product2d_s(a,b);
        case 3
            c = product3d_s(a,b);
        case 4
            c = product4d_s(a,b);
    end
end

function display
    fprintf('Function: %s\n',mfilename);
    fprintf('Space dimension: %d\n',n);
    fprintf('Algebra dimension: %d\n',d);
    fprintf('Basis multivectors:\n');
    switch n
        case 1
            fprintf('\te(1) = 1 - scalar\n');
            fprintf('\te(2) = e1 - vector (or pseudoscalar)\n');
        case 2
            fprintf('\te(1) = 1 - scalar\n');
            fprintf('\te(2) = e1 - vector 1\n');
            fprintf('\te(3) = e2 - vector 2\n');
            fprintf('\te(4) = e12 - pseudoscalar\n');
        case 3
            fprintf('\te(1) = 1 - scalar\n');
            fprintf('\te(2) = e1 - vector 1\n');
            fprintf('\te(3) = e2 - vector 2\n');
            fprintf('\te(4) = e3 - vector 3\n');
    end
end

```

```

        fprintf('\te(5) = e12 - bi-vector 1\n');
        fprintf('\te(6) = e13 - bi-vector 2\n');
        fprintf('\te(7) = e23 - bi-vector 3\n');
        fprintf('\te(8) = e123 - pseudoscalar\n');
    case 4
        fprintf('\te(1) = 1 - scalar\n');
        fprintf('\te(2) = e1 - vector 1\n');
        fprintf('\te(3) = e2 - vector 2\n');
        fprintf('\te(4) = e3 - vector 3\n');
        fprintf('\te(5) = e4 - vector 4\n');
        fprintf('\te(6) = e12 - bi-vector 1\n');
        fprintf('\te(7) = e13 - bi-vector 2\n');
        fprintf('\te(8) = e14 - bi-vector 3\n');
        fprintf('\te(9) = e23 - bi-vector 4\n');
        fprintf('\te(10) = e24 - bi-vector 5\n');
        fprintf('\te(11) = e34 - bi-vector 6\n');
        fprintf('\te(12) = e123 - tri-vector 1\n');
        fprintf('\te(13) = e124 - tri-vector 2\n');
        fprintf('\te(14) = e134 - tri-vector 3\n');
        fprintf('\te(15) = e234 - tri-vector 4\n');
        fprintf('\te(16) = e1234 - pseudoscalar\n');
    end
end

CA.dimension = d;
CA.set_scalar = @set_scalar;
CA.get_scalar = @get_scalar;
CA.set_vector = @set_vector;
CA.get_vector = @get_vector;
CA.even = @even;
CA.reverse = @reverse;
CA.dual = @dual;
CA.dual_inv = @dual_inv;
CA.product = @product;
CA.product_p = @product_p;
CA.product_r = @product_r;
CA.product_s = @product_s;
CA.display = @display;
end

%=====
function c = dual1d(c)
c0 = c(:,1);
c1 = c(:,2);
c = [c1,c0];
end

%=====
function c = product1d(a,b)
a0 = a(:,1);
a1 = a(:,2);
b0 = b(:,1);
b1 = b(:,2);
c0 = a0.*b0 + a1.*b1;
c1 = a0.*b1 + a1.*b0;
c = [c0,c1];
end

%=====
function c = product1d_p(a,b)
a0 = a(:,1);
a1 = a(:,2);

b0 = b(:,1);
b1 = b(:,2);

c0 = a0.*b0;

```

```

c1 = a0.*b1 + a1.*b0;
c = [c0,c1];
end

%=====
function c = product1d_r(a,b)
a = dual1d(a);
b = dual1d(b);
c = product1d_p(a,b);
c = dual1d(c);
end

%=====
function c = dual2d(c)
c0 = c(:,1);
c1 = c(:,2);
c2 = c(:,3);
c12 = c(:,4);
c = [c12,c2,-c1,c0];
end

%=====
function c = dual2d_inv(c)
c0 = c(:,1);
c1 = c(:,2);
c2 = c(:,3);
c12 = c(:,4);
c = [c12,-c2,c1,c0];
end

%=====
function c = product2d(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a12 = a(:,4);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b12 = b(:,4);

c0 = a0.*b0 + a1.*b1 + a2.*b2 - a12.*b12;
c1 = a0.*b1 + a1.*b0 - a2.*b12 + a12.*b2;
c2 = a0.*b2 + a1.*b12 + a2.*b0 - a12.*b1;
c12 = a0.*b12 + a1.*b2 - a2.*b1 + a12.*b0;

c = [c0,c1,c2,c12];
end

%=====
function c = product2d_p(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a12 = a(:,4);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b12 = b(:,4);

c0 = a0.*b0;
c1 = a0.*b1 + a1.*b0;
c2 = a0.*b2 + a2.*b0;
c12 = a0.*b12 + a1.*b2 - a2.*b1 + a12.*b0;

```

```

c = [c0,c1,c2,c12];
end

%=====
function c = product2d_r(a,b)
a = dual2d(a);
b = dual2d(b);
c = product2d_p(a,b);
c = dual2d_inv(c);
end

%=====
function c = product2d_s(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a12 = a(:,4);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b12 = b(:,4);

c0 = a0.*b0 + a1.*b1 + a2.*b2 - a12.*b12;
c1 = a0.*b1 + a1.*b0 - a2.*b12 + a12.*b2;
c2 = a0.*b2 + a1.*b12 + a2.*b0 - a12.*b1;
c12 = a0.*b12 + a12.*b0;

c = [c0,c1,c2,c12];
end

%=====
function c = dual3d(c)
c0 = c(:,1);
c1 = c(:,2);
c2 = c(:,3);
c3 = c(:,4);
c12 = c(:,5);
c13 = c(:,6);
c23 = c(:,7);
c123 = c(:,8);
c = [c123,c23,-c13,c12,c3,-c2,c1,c0];
end

%=====
function c = product3d(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a3 = a(:,4);
a12 = a(:,5);
a13 = a(:,6);
a23 = a(:,7);
a123 = a(:,8);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b3 = b(:,4);
b12 = b(:,5);
b13 = b(:,6);
b23 = b(:,7);
b123 = b(:,8);

c0 = a0.*b0 ...
    + a1.*b1 + a2.*b2 + a3.*b3 ...
    - a12.*b12 - a13.*b13 - a23.*b23 ...

```

```

    - a123.*b123;
c1 = a0.*b1 ...
    + a1.*b0 - a2.*b12 - a3.*b13 ...
    + a12.*b2 + a13.*b3 - a23.*b123 ...
    - a123.*b23;
c2 = a0.*b2 ...
    + a1.*b12 + a2.*b0 - a3.*b23 ...
    - a12.*b1 + a13.*b123 + a23.*b3 ...
    + a123.*b13;
c3 = a0.*b3 ...
    + a1.*b13 + a2.*b23 + a3.*b0 ...
    - a12.*b123 - a13.*b1 - a23.*b2 ...
    - a123.*b12;
c12 = a0.*b12 ...
    + a1.*b2 - a2.*b1 + a3.*b123 ...
    + a12.*b0 - a13.*b23 + a23.*b13 ...
    + a123.*b3;
c13 = a0.*b13 ...
    + a1.*b3 - a2.*b123 - a3.*b1 ...
    + a12.*b23 + a13.*b0 - a23.*b12 ...
    - a123.*b2;
c23 = a0.*b23 ...
    + a1.*b123 + a2.*b3 - a3.*b2 ...
    - a12.*b13 + a13.*b12 + a23.*b0 ...
    + a123.*b1;
c123 = a0.*b123 ...
    + a1.*b23 - a2.*b13 + a3.*b12 ...
    + a12.*b3 - a13.*b2 + a23.*b1 ...
    + a123.*b0;

c = [c0,c1,c2,c3,c12,c13,c23,c123];
end

%=====
function c = product3d_p(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a3 = a(:,4);
a12 = a(:,5);
a13 = a(:,6);
a23 = a(:,7);
a123 = a(:,8);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b3 = b(:,4);
b12 = b(:,5);
b13 = b(:,6);
b23 = b(:,7);
b123 = b(:,8);

c0 = a0.*b0;
c1 = a0.*b1 + a1.*b0;
c2 = a0.*b2 + a2.*b0;
c3 = a0.*b3 + a3.*b0;
c12 = a0.*b12 + a1.*b2 - a2.*b1 + a12.*b0;
c13 = a0.*b13 + a1.*b3 - a3.*b1 + a13.*b0;
c23 = a0.*b23 + a2.*b3 - a3.*b2 + a23.*b0;
c123 = a0.*b123 ...
    + a1.*b23 - a2.*b13 + a3.*b12 ...
    + a12.*b3 - a13.*b2 + a23.*b1 ...
    + a123.*b0;

c = [c0,c1,c2,c3,c12,c13,c23,c123];
end

```

```

%=====
function c = product3d_r(a,b)
a = dual3d(a);
b = dual3d(b);
c = product3d_p(a,b);
c = dual3d(c);
end

%=====
function c = product3d_s(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a3 = a(:,4);
a12 = a(:,5);
a13 = a(:,6);
a23 = a(:,7);
a123 = a(:,8);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b3 = b(:,4);
b12 = b(:,5);
b13 = b(:,6);
b23 = b(:,7);
b123 = b(:,8);

c0 = a0.*b0 + a1.*b1 + a2.*b2 + a3.*b3 ...
    - a12.*b12 - a13.*b13 - a23.*b23 - a123.*b123;
c1 = a0.*b1 + a1.*b0 - a2.*b12 - a3.*b13 ...
    + a12.*b2 + a13.*b3 - a23.*b123 - a123.*b23;
c2 = a0.*b2 + a1.*b12 + a2.*b0 - a3.*b23 ...
    - a12.*b1 + a13.*b123 + a23.*b3 + a123.*b13;
c3 = a0.*b3 + a1.*b13 + a2.*b23 + a3.*b0 ...
    - a12.*b123 - a13.*b1 - a23.*b2 - a123.*b12;
c12 = a0.*b12 + a3.*b123 + a12.*b0 + a123.*b3;
c13 = a0.*b13 - a2.*b123 + a13.*b0 - a123.*b2;
c23 = a0.*b23 + a1.*b123 + a23.*b0 + a123.*b1;
c123 = a0.*b123 + a123.*b0;

c = [c0,c1,c2,c3,c12,c13,c23,c123];
end

%=====
function c = dual4d(c)
c0 = c(:,1);
c1 = c(:,2);
c2 = c(:,3);
c3 = c(:,4);
c4 = c(:,5);
c12 = c(:,6);
c13 = c(:,7);
c14 = c(:,8);
c23 = c(:,9);
c24 = c(:,10);
c34 = c(:,11);
c123 = c(:,12);
c124 = c(:,13);
c134 = c(:,14);
c234 = c(:,15);
c1234 = c(:,16);
c = [c1234,...
    c234,-c134,c124,-c123,...
    c34,-c24,c23,c14,-c13,c12,...
    c4,-c3,c2,-c1,...

```

```

    c0];
end

%=====
function c = dual4d_inv(c)
c0 = c(:,1);
c1 = c(:,2);
c2 = c(:,3);
c3 = c(:,4);
c4 = c(:,5);
c12 = c(:,6);
c13 = c(:,7);
c14 = c(:,8);
c23 = c(:,9);
c24 = c(:,10);
c34 = c(:,11);
c123 = c(:,12);
c124 = c(:,13);
c134 = c(:,14);
c234 = c(:,15);
c1234 = c(:,16);
c = [c1234,...
     -c234,c134,-c124,c123,...
     c34,-c24,c23,c14,-c13,c12,...
     -c4,c3,-c2,c1,...
     c0];
end

%=====
function c = product4d(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a3 = a(:,4);
a4 = a(:,5);
a12 = a(:,6);
a13 = a(:,7);
a14 = a(:,8);
a23 = a(:,9);
a24 = a(:,10);
a34 = a(:,11);
a123 = a(:,12);
a124 = a(:,13);
a134 = a(:,14);
a234 = a(:,15);
a1234 = a(:,16);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b3 = b(:,4);
b4 = b(:,5);
b12 = b(:,6);
b13 = b(:,7);
b14 = b(:,8);
b23 = b(:,9);
b24 = b(:,10);
b34 = b(:,11);
b123 = b(:,12);
b124 = b(:,13);
b134 = b(:,14);
b234 = b(:,15);
b1234 = b(:,16);

c0 = a0.*b0 ...
    + a1.*b1 + a2.*b2 + a3.*b3 + a4.*b4 ...
    - a12.*b12 - a13.*b13 - a14.*b14 - a23.*b23 - a24.*b24 - a34.*b34 ...

```

```

- a123.*b123 - a124.*b124 - a134.*b134 - a234.*b234 ...
+ a1234.*b1234;
c1 = a0.*b1 + a1.*b0 - a2.*b12 - a3.*b13 - a4.*b14 ...
+ a12.*b2 + a13.*b3 + a14.*b4 - a23.*b123 - a24.*b124 - a34.*b134 ...
- a123.*b23 - a124.*b24 - a134.*b34 + a234.*b1234 ...
- a1234.*b234;
c2 = a0.*b2 ...
+ a1.*b12 + a2.*b0 - a3.*b23 - a4.*b24 ...
- a12.*b1 + a13.*b123 + a14.*b124 + a23.*b3 + a24.*b4 - a34.*b234 ...
+ a123.*b13 + a124.*b14 - a134.*b1234 - a234.*b34 ...
+ a1234.*b134;
c3 = a0.*b3 ...
+ a1.*b13 + a2.*b23 + a3.*b0 - a4.*b34 ...
- a12.*b123 - a13.*b1 + a14.*b134 - a23.*b2 + a24.*b234 + a34.*b4 ...
- a123.*b12 + a124.*b1234 + a134.*b14 + a234.*b24 ...
- a1234.*b124;
c4 = a0.*b4 ...
+ a1.*b14 + a2.*b24 + a3.*b34 + a4.*b0 ...
- a12.*b124 - a13.*b134 - a14.*b1 - a23.*b234 - a24.*b2 - a34.*b3 ...
- a123.*b1234 - a124.*b12 - a134.*b13 - a234.*b23 ...
+ a1234.*b123;
c12 = a0.*b12 ...
+ a1.*b2 - a2.*b1 + a3.*b123 + a4.*b124 ...
+ a12.*b0 - a13.*b23 - a14.*b24 + a23.*b13 + a24.*b14 - a34.*b1234 ...
+ a123.*b3 + a124.*b4 - a134.*b234 + a234.*b134 ...
- a1234.*b34;
c13 = a0.*b13 ...
+ a1.*b3 - a2.*b123 - a3.*b1 + a4.*b134 ...
+ a12.*b23 + a13.*b0 - a14.*b34 - a23.*b12 + a24.*b1234 + a34.*b14 ...
- a123.*b2 + a124.*b234 + a134.*b4 - a234.*b124 ...
+ a1234.*b24;
c14 = a0.*b14 ...
+ a1.*b4 - a2.*b124 - a3.*b134 - a4.*b1 ...
+ a12.*b24 + a13.*b34 + a14.*b0 - a23.*b1234 - a24.*b12 - a34.*b13 ...
- a123.*b234 - a124.*b2 - a134.*b3 + a234.*b123 ...
- a1234.*b23;
c23 = a0.*b23 ...
+ a1.*b123 + a2.*b3 - a3.*b2 + a4.*b234 ...
- a12.*b13 + a13.*b12 - a14.*b1234 + a23.*b0 - a24.*b34 + a34.*b24 ...
+ a123.*b1 - a124.*b134 + a134.*b124 + a234.*b4 ...
- a1234.*b14;
c24 = a0.*b24 ...
+ a1.*b124 + a2.*b4 - a3.*b234 - a4.*b2 ...
- a12.*b14 + a13.*b1234 + a14.*b12 + a23.*b34 + a24.*b0 - a34.*b23 ...
+ a123.*b134 + a124.*b1 - a134.*b123 - a234.*b3 ...
+ a1234.*b13;
c34 = a0.*b34 ...
+ a1.*b134 + a2.*b234 + a3.*b4 - a4.*b3 ...
- a12.*b1234 - a13.*b14 + a14.*b13 - a23.*b24 + a24.*b23 + a34.*b0 ...
- a123.*b124 + a124.*b123 + a134.*b1 + a234.*b2 ...
- a1234.*b12;
c123 = a0.*b123 ...
+ a1.*b23 - a2.*b13 + a3.*b12 - a4.*b1234 ...
+ a12.*b3 - a13.*b2 + a14.*b234 + a23.*b1 - a24.*b134 + a34.*b124 ...
+ a123.*b0 - a124.*b34 + a134.*b24 - a234.*b14 ...
+ a1234.*b4;
c124 = a0.*b124 ...
+ a1.*b24 - a2.*b14 + a3.*b1234 + a4.*b12 ...
+ a12.*b4 - a13.*b234 - a14.*b2 + a23.*b134 + a24.*b1 - a34.*b123 ...
+ a123.*b34 + a124.*b0 - a134.*b23 + a234.*b13 ...
- a1234.*b3;
c134 = a0.*b134 ...
+ a1.*b34 - a2.*b1234 - a3.*b14 + a4.*b13 ...
+ a12.*b234 + a13.*b4 - a14.*b3 - a23.*b124 + a24.*b123 + a34.*b1 ...
- a123.*b24 + a124.*b23 + a134.*b0 - a234.*b12 ...
+ a1234.*b2;
c234 = a0.*b234 ...

```

```

+ a1.*b1234 + a2.*b34 - a3.*b24 + a4.*b23 ...
- a12.*b134 + a13.*b124 - a14.*b123 + a23.*b4 - a24.*b3 + a34.*b2 ...
+ a123.*b14 - a124.*b13 + a134.*b12 + a234.*b0 ...
- a1234.*b1;
c1234 = a0.*b1234 ...
+ a1.*b234 - a2.*b134 + a3.*b124 - a4.*b123 ...
+ a12.*b34 - a13.*b24 + a14.*b23 + a23.*b14 - a24.*b13 + a34.*b12 ...
+ a123.*b4 - a124.*b3 + a134.*b2 - a234.*b1 + a1234.*b0;

c = [c0,c1,c2,c3,c4,c12,c13,c14,c23,c24,c34,c123,c124,c134,c234,c1234];
end

%=====
function c = product4d_p(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a3 = a(:,4);
a4 = a(:,5);
a12 = a(:,6);
a13 = a(:,7);
a14 = a(:,8);
a23 = a(:,9);
a24 = a(:,10);
a34 = a(:,11);
a123 = a(:,12);
a124 = a(:,13);
a134 = a(:,14);
a234 = a(:,15);
a1234 = a(:,16);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b3 = b(:,4);
b4 = b(:,5);
b12 = b(:,6);
b13 = b(:,7);
b14 = b(:,8);
b23 = b(:,9);
b24 = b(:,10);
b34 = b(:,11);
b123 = b(:,12);
b124 = b(:,13);
b134 = b(:,14);
b234 = b(:,15);
b1234 = b(:,16);

c0 = a0.*b0;
c1 = a0.*b1 + a1.*b0;
c2 = a0.*b2 + a2.*b0;
c3 = a0.*b3 + a3.*b0;
c4 = a0.*b4 + a4.*b0;
c12 = a0.*b12 + a1.*b2 - a2.*b1 + a12.*b0;
c13 = a0.*b13 + a1.*b3 - a3.*b1 + a13.*b0;
c14 = a0.*b14 + a1.*b4 - a4.*b1 + a14.*b0;
c23 = a0.*b23 + a2.*b3 - a3.*b2 + a23.*b0;
c24 = a0.*b24 + a2.*b4 - a4.*b2 + a24.*b0;
c34 = a0.*b34 + a3.*b4 - a4.*b3 + a34.*b0;
c123 = a0.*b123 + a1.*b23 - a2.*b13 + a3.*b12 ...
+ a12.*b3 - a13.*b2 + a23.*b1 + a123.*b0;
c124 = a0.*b124 + a1.*b24 - a2.*b14 + a4.*b12 ...
+ a12.*b4 - a14.*b2 + a24.*b1 + a124.*b0;
c134 = a0.*b134 + a1.*b34 - a3.*b14 + a4.*b13 ...
+ a13.*b4 - a14.*b3 + a34.*b1 + a134.*b0;
c234 = a0.*b234 + a2.*b34 - a3.*b24 + a4.*b23 ...
+ a23.*b4 - a24.*b3 + a34.*b2 + a234.*b0;

```

```

c1234 = a0.*b1234 ...
    + a1.*b234 - a2.*b134 + a3.*b124 - a4.*b123 ...
    + a12.*b34 - a13.*b24 + a14.*b23 + a23.*b14 - a24.*b13 + a34.*b12 ...
    + a123.*b4 - a124.*b3 + a134.*b2 - a234.*b1 + a1234.*b0;

c = [c0,c1,c2,c3,c4,c12,c13,c14,c23,c24,c34,c123,c124,c134,c234,c1234];
end

%=====
function c = product4d_r(a,b)
a = dual4d(a);
b = dual4d(b);
c = product4d_p(a,b);
c = dual4d_inv(c);
end

%=====
function c = product4d_s(a,b)
a0 = a(:,1);
a1 = a(:,2);
a2 = a(:,3);
a3 = a(:,4);
a4 = a(:,5);
a12 = a(:,6);
a13 = a(:,7);
a14 = a(:,8);
a23 = a(:,9);
a24 = a(:,10);
a34 = a(:,11);
a123 = a(:,12);
a124 = a(:,13);
a134 = a(:,14);
a234 = a(:,15);
a1234 = a(:,16);

b0 = b(:,1);
b1 = b(:,2);
b2 = b(:,3);
b3 = b(:,4);
b4 = b(:,5);
b12 = b(:,6);
b13 = b(:,7);
b14 = b(:,8);
b23 = b(:,9);
b24 = b(:,10);
b34 = b(:,11);
b123 = b(:,12);
b124 = b(:,13);
b134 = b(:,14);
b234 = b(:,15);
b1234 = b(:,16);

c0 = a0.*b0 ...
    + a1.*b1 + a2.*b2 + a3.*b3 + a4.*b4 ...
    - a12.*b12 - a13.*b13 - a14.*b14 - a23.*b23 - a24.*b24 - a34.*b34 ...
    - a123.*b123 - a124.*b124 - a134.*b134 - a234.*b234 ...
    + a1234.*b1234;
c1 = a0.*b1 + a1.*b0 - a2.*b12 - a3.*b13 - a4.*b14 ...
    + a12.*b2 + a13.*b3 + a14.*b4 - a23.*b123 - a24.*b124 - a34.*b134 ...
    - a123.*b23 - a124.*b24 - a134.*b34 + a234.*b1234 ...
    - a1234.*b234;
c2 = a0.*b2 ...
    + a1.*b12 + a2.*b0 - a3.*b23 - a4.*b24 ...
    - a12.*b1 + a13.*b123 + a14.*b124 + a23.*b3 + a24.*b4 - a34.*b234 ...
    + a123.*b13 + a124.*b14 - a134.*b1234 - a234.*b34 ...
    + a1234.*b134;
c3 = a0.*b3 ...

```

```

+ a1.*b13 + a2.*b23 + a3.*b0 - a4.*b34 ...
- a12.*b123 - a13.*b1 + a14.*b134 - a23.*b2 + a24.*b234 + a34.*b4 ...
- a123.*b12 + a124.*b1234 + a134.*b14 + a234.*b24 ...
- a1234.*b124;
c4 = a0.*b4 ...
+ a1.*b14 + a2.*b24 + a3.*b34 + a4.*b0 ...
- a12.*b124 - a13.*b134 - a14.*b1 - a23.*b234 - a24.*b2 - a34.*b3 ...
- a123.*b1234 - a124.*b12 - a134.*b13 - a234.*b23 ...
+ a1234.*b123;
c12 = a0.*b12 + a3.*b123 + a4.*b124 + a12.*b0 - a34.*b1234 ...
+ a123.*b3 + a124.*b4 - a1234.*b34;
c13 = a0.*b13 - a2.*b123 + a4.*b134 + a13.*b0 + a24.*b1234 ...
- a123.*b2 + a134.*b4 + a1234.*b24;
c14 = a0.*b14 - a2.*b124 - a3.*b134 + a14.*b0 - a23.*b1234 ...
- a124.*b2 - a134.*b3 - a1234.*b23;
c23 = a0.*b23 + a1.*b123 + a4.*b234 - a14.*b1234 + a23.*b0 ...
+ a123.*b1 + a234.*b4 - a1234.*b14;
c24 = a0.*b24 + a1.*b124 - a3.*b234 + a13.*b1234 + a24.*b0 ...
+ a124.*b1 - a234.*b3 + a1234.*b13;
c34 = a0.*b34 + a1.*b134 + a2.*b234 - a12.*b1234 + a34.*b0 ...
+ a134.*b1 + a234.*b2 - a1234.*b12;
c123 = a0.*b123 - a4.*b1234 + a123.*b0 + a1234.*b4;
c124 = a0.*b124 + a3.*b1234 + a124.*b0 - a1234.*b3;
c134 = a0.*b134 - a2.*b1234 + a134.*b0 + a1234.*b2;
c234 = a0.*b234 + a1.*b1234 + a234.*b0 - a1234.*b1;
c1234 = a0.*b1234 + a1234.*b0;

c = [c0,c1,c2,c3,c4,c12,c13,c14,c23,c24,c34,c123,c124,c134,c234,c1234];
end

```

Appendix C Listing of clifford_algebra_test.m

```
%CLIFFORD_ALGEBRA_TEST(N,tol)
%
%Unit tests of basic operations of Clifford algebra.
%
%Copyright (C) 2014 Defence Science and Technology Organisation
%
%Created by Leonid K. Antanovskii
function clifford_algebra_test(N,tol)
fprintf('Testing Clifford algebra\n');
if nargin < 2
    rng('default');
    tol = 1.0e-14;
    if nargin < 1
        N = 100000;
    end
end

for n = 1:4
    fprintf('Space dimension: %d\n',n);
    test_geometric_product(n,tol,N);
    test_progressive_product(n,tol,N);
    test_regressive_product(n,tol,N);
    test_scalar_product(n,tol,N);
    test_inner_product(n,tol,N);
    test_even_subalgebra(n,N);
    test_reverse_operator(n,tol,N);
    test_dual_operator(n,tol,N);
    test_blade_feature(n,tol,N);
    test_identity_1(n,tol,N);
    test_identity_2(n,tol,N);
    test_identity_3(n,tol,N);
    test_identity_4(n,tol,N);
    test_identity_5(n,tol);
end

test_cross_product(tol,N);
test_complex_number(tol,N);
test_central_subalgebra(tol,N);
test_bicomplex_number(tol,N);
test_quaternion(tol,N);
test_performance(N);

end

%=====
function test_geometric_product(n,tol,N)
fprintf('Geometric product test\n');

CA = clifford_algebra(n);
d = CA.dimension;

% random multivectors
a = 2.0*rand(N,d) - 1.0;
b = 2.0*rand(N,d) - 1.0;
c = 2.0*rand(N,d) - 1.0;

% check associativity
q = CA.product(CA.product(a,b),c) - CA.product(a,CA.product(b,c));
assert(all(q(:) < tol));
end

%=====
function test_progressive_product(n,tol,N)
fprintf('Progressive outer product test\n');
```

```

CA = clifford_algebra(n);
d = CA.dimension;

% random multivectors
a = 2.0*rand(N,d) - 1.0;
b = 2.0*rand(N,d) - 1.0;
c = 2.0*rand(N,d) - 1.0;

% check associativity
q = CA.product_p(CA.product_p(a,b),c) - CA.product_p(a,CA.product_p(b,c));
assert(all(q(:) < tol));
end

%=====
function test_regressive_product(n,tol,N)
fprintf('Regressive outer product test\n');

CA = clifford_algebra(n);
d = CA.dimension;

% random multivectors
a = 2.0*rand(N,d) - 1.0;
b = 2.0*rand(N,d) - 1.0;
c = 2.0*rand(N,d) - 1.0;

% check associativity
q = CA.product_r(CA.product_r(a,b),c) - CA.product_r(a,CA.product_r(b,c));
assert(all(q(:) < tol));
end

%=====
function test_scalar_product(n,tol,N)
fprintf('Scalar product test\n');

CA = clifford_algebra(n);

a = 2.0*rand(N,n) - 1.0;
b = 2.0*rand(N,n) - 1.0;
ab1 = CA.set_scalar(dot(a,b,2));

a = CA.set_vector(a);
b = CA.set_vector(b);
ab2 = CA.product_s(a,b);

assert(norm(ab1 - ab2,inf) < tol);
end

%=====
function test_inner_product(n,tol,N)
fprintf('Inner product test\n');

CA = clifford_algebra(n);

for k = 1:n
    fprintf('\twedge product of %d vectors\n',k);

    a = CA.set_vector(2.0*rand(N,n) - 1.0);
    b = cell(k,1);
    for i = 1:k
        b{i} = CA.set_vector(2.0*rand(N,n) - 1.0);
    end

    p = b{1};
    for i = 2:k
        p = CA.product_p(p,b{i});
    end
end

```

```

    p = CA.product_s(a,p);

    q = CA.set_vector(zeros(N,n));
    for i = 1:k
        s = CA.product_s(a,b{i});
        for j = 1:k
            if j ~= i
                s = CA.product_p(s,b{j});
            end
        end
        q = q + (-1)^(i-1)*s;
    end

    assert(norm(p - q,inf) < tol);
end
end

%=====
function test_even_subalgebra(n,N)
fprintf('Even subalgebra test\n');

CA = clifford_algebra(n);
d = CA.dimension;

% random even multivectors
a = CA.even(2.0*rand(N,d) - 1.0);
b = CA.even(2.0*rand(N,d) - 1.0);

% check subalgebra property
c = CA.product(a,b);
assert(norm(CA.even(c) - c,inf) == 0.0);
end

%=====
function test_reverse_operator(n,tol,N)
fprintf('Reversion operator test\n');

CA = clifford_algebra(n);
d = CA.dimension;

% random multivectors
a = 2.0*rand(N,d) - 1.0;

% check involution
b = CA.reverse(a);
c = CA.reverse(b);
assert(norm(a - c,inf) < tol);
end

%=====
function test_dual_operator(n,tol,N)
fprintf('Dual operator test\n');

CA = clifford_algebra(n);
d = CA.dimension;

% random multivectors
a = 2.0*rand(N,d) - 1.0;

% check inversion
b = CA.dual(a);
c = CA.dual_inv(b);
assert(norm(a - c,inf) < tol);
end

%=====
function test_blade_feature(n,tol,N)

```

```

fprintf('Blade feature test\n');

CA = clifford_algebra(n);
e = CA.set_scalar(1.0); % unit
v = orth(2.0*rand(n,n) - 1.0); % random orthonormal basis

for k = 1:n
    fprintf('\t%d-blade\n',k);

    % construct a blade
    A = e;
    for i = 1:k
        a = CA.set_vector(v(i,:));
        A = CA.product(A,a);
    end

    % random vectors
    a = CA.set_vector(2.0*rand(N,n) - 1.0);

    % check blade properties
    p = CA.product(a,A) - CA.product_s(a,A) - CA.product_p(a,A);
    assert(all(p(:) < tol));
    p = CA.product(A,a) - CA.product_s(A,a) - CA.product_p(A,a);
    assert(all(p(:) < tol));
end

for k = 1:n-1
    fprintf('\t%d-blade and complementary %d-blade\n',k,n-k);

    % construct complementary blades
    A = e;
    A_inv = e;
    for i = 1:k
        a = CA.set_vector(v(i,:));
        A = CA.product(A,a);
        A_inv = CA.product(a,A_inv);
    end
    B = CA.dual(A);
    V = CA.product(A,B); % must be a volume form (pseudoscalar)

    % check blade inversion
    assert(norm(CA.product(A,A_inv) - e,inf) < tol);
    assert(norm(CA.product(A_inv,A) - e,inf) < tol);

    % check compatibility with reversion
    assert(norm(CA.reverse(A) - A_inv,inf) < tol);

    % check blade orthogonality
    assert(norm(CA.product_s(A,B),inf) < tol);
    assert(norm(CA.product_p(A,B) - V,inf) < tol);

    % random vectors
    a = CA.set_vector(2.0*rand(N,n) - 1.0);

    % projection and rejection
    p = CA.product(CA.product_s(a,A),A_inv);
    r = CA.product(CA.product_p(a,A),A_inv);

    % check identities
    assert(norm(p + r - a,inf) < tol); % sum decomposition
    assert(norm(CA.product_s(p,r),inf) < tol); % orthogonality
    assert(norm(CA.product_p(p,A),inf) < tol); % inclusion of projected
    assert(norm(CA.product_p(r,B),inf) < tol); % inclusion of rejected
    assert(norm(CA.product_s(p,B),inf) < tol); % normality of projected
    assert(norm(CA.product_s(r,A),inf) < tol); % normality of rejected
    assert(norm(CA.product_p(a,V),inf) < tol); % complete vector space
end

```

```

end

%=====
function test_identity_1(n,tol,N)
fprintf('Test of identity:  $a^b = -b^a$ \n');

CA = clifford_algebra(n);

% random vectors
a = CA.set_vector(2.0*rand(N,n) - 1.0);
b = CA.set_vector(2.0*rand(N,n) - 1.0);

% check antisymmetry
q = CA.product_p(a,b) + CA.product_p(b,a);
assert(all(q(:) < tol));
end

%=====
function test_identity_2(n,tol,N)
fprintf('Test of identity:  $a.(b^c) = (a.b)c - (a.c)b$ \n');

CA = clifford_algebra(n);

% random vectors
a = CA.set_vector(2.0*rand(N,n) - 1.0);
b = CA.set_vector(2.0*rand(N,n) - 1.0);
c = CA.set_vector(2.0*rand(N,n) - 1.0);

% check the identity
abc1 = CA.product_s(a,CA.product_p(b,c));
abc2 = CA.product(CA.product_s(a,b),c) - CA.product(CA.product_s(a,c),b);
assert(norm(abc1 - abc2,inf) < tol);
end

%=====
function test_identity_3(n,tol,N)
fprintf('Test of identity:  $(a^b).(c^d) = (a.d)(b.c) - (a.c)(b.d)$ \n');

CA = clifford_algebra(n);

% random vectors
a = CA.set_vector(2.0*rand(N,n) - 1.0);
b = CA.set_vector(2.0*rand(N,n) - 1.0);
c = CA.set_vector(2.0*rand(N,n) - 1.0);
d = CA.set_vector(2.0*rand(N,n) - 1.0);

% check the identity
ab = CA.product_p(a,b);
cd = CA.product_p(c,d);
ab_cd = CA.product_s(ab,cd);
a_d = CA.product_s(a,d);
b_c = CA.product_s(b,c);
a_c = CA.product_s(a,c);
b_d = CA.product_s(b,d);
assert(norm(ab_cd - (a_d.*b_c - a_c.*b_d),inf) < tol);
end

%=====
function test_identity_4(n,tol,N)
fprintf('Test of identity:  $a^C b = -b^C a$ \n');

CA = clifford_algebra(n);
d = CA.dimension;

% random vectors
a = CA.set_vector(2.0*rand(N,n) - 1.0);
b = CA.set_vector(2.0*rand(N,n) - 1.0);

```

```

% random multivectors
c = 2.0*rand(N,d) - 1.0;

% check antisymmetry
acb = CA.product_p(a,CA.product_p(c,b));
bca = CA.product_p(b,CA.product_p(c,a));
assert(norm(acb + bca,inf) < tol);
end

%=====
function test_identity_5(n,tol)
fprintf('Test of identity: a_1^...^a_n = det(a)e_{1...n}\n');

CA = clifford_algebra(n);
d = CA.dimension;

% random basis vectors
v = 2.0*rand(n,n) - 1.0;

% check the identity
p = CA.set_scalar(1.0);
for i = 1:n
    p = CA.product_p(p,CA.set_vector(v(i,:)));
end
q = det(v);
assert(abs(p(d) - q) < tol);
end

%=====
function test_cross_product(tol,N)
fprintf('Test of cross product: a x b = *(a^b)\n');

CA = clifford_algebra(3);

% random vectors
a = 2.0*rand(N,3) - 1.0;
b = 2.0*rand(N,3) - 1.0;

% check the identity
ab1 = CA.set_vector(cross(a,b,2));
a = CA.set_vector(a);
b = CA.set_vector(b);
ab2 = CA.dual(CA.product_p(a,b));
assert(norm(ab1 - ab2,inf) < tol);
end

%=====
function test_complex_number(tol,N)
fprintf('Complex number test\n');

CA = clifford_algebra(3);

% random complex numbers
z1 = 2.0*rand(N,2) - 1.0;
c1 = [z1(:,1),zeros(N,6),z1(:,2)];
z1 = complex(z1(:,1),z1(:,2));
z2 = 2.0*rand(N,2) - 1.0;
c2 = [z2(:,1),zeros(N,6),z2(:,2)];
z2 = complex(z2(:,1),z2(:,2));

% check the products
c = CA.product(c1,c2);
z = z1.*z2;
cz = [real(z),zeros(N,6),imag(z)];
assert(norm(cz - c,inf) < tol);
end

```

```

%=====
function test_central_subalgebra(tol,N)
fprintf('Central subalgebra test\n');

CA = clifford_algebra(3);

% random complex numbers
z = 2.0*rand(N,2) - 1.0;
z = [z(:,1),zeros(N,6),z(:,2)];

% random multivectors
c = 2.0*rand(N,8) - 1.0;

% check commutativity
zc = CA.product(z,c);
cz = CA.product(c,z);
assert(norm(zc - cz,inf) < tol);
end

%=====
function test_bicomplex_number(tol,N)
fprintf('Bicomplex number test\n');

CA = clifford_algebra(3);

% random bicomplex numbers
a = 2.0*rand(N,4) - 1.0;
a = [a(:,1:2),zeros(N,4),a(:,3:4)];
b = 2.0*rand(N,4) - 1.0;
b = [b(:,1:2),zeros(N,4),b(:,3:4)];

% check commutativity
q = CA.product(a,b) - CA.product(b,a);
assert(all(q(:) < tol));
end

%=====
function test_quaternion(tol,N)
fprintf('Quaternion test\n');

CA = clifford_algebra(3);
e = CA.set_scalar(ones(N,1)); % units

% random quaternions and their inverses
q = CA.even(2.0*rand(N,8) - 1.0);
q_inv = CA.reverse(q);
s = CA.product_s(q,q_inv);
s(:,1) = 1.0./s(:,1);
q_inv = CA.product(s,q_inv);

% check quaternion inversion
assert(norm(CA.product(q,q_inv) - e,inf) < tol);
assert(norm(CA.product(q_inv,q) - e,inf) < tol);

% random vectors and their vector/scalar products
a = CA.set_vector(2.0*rand(N,3) - 1.0);
b = CA.set_vector(2.0*rand(N,3) - 1.0);
c = CA.dual(CA.product_p(a,b));
s = CA.product_s(a,b);

% rotated vectors and their vector/scalar products
a1 = CA.product(q,CA.product(a,q_inv));
b1 = CA.product(q,CA.product(b,q_inv));
c1 = CA.dual(CA.product_p(a1,b1));
s1 = CA.product_s(a1,b1);

```

```

% check space invariance
a2 = CA.set_vector(CA.get_vector(a1));
assert(norm(a2 - a1,inf) < tol);
b2 = CA.set_vector(CA.get_vector(b1));
assert(norm(b2 - b1,inf) < tol);

% check isometric property
assert(norm(s1 - s,inf) < tol);

% check orientation preserving property
c2 = CA.product(q,CA.product(c,q_inv));
assert(norm(c2 - c1,inf) < tol);
end

%=====
function test_performance(N)
fprintf('Performance test\n');

CA = clifford_algebra(4);
CA.display();
d = CA.dimension;

% random multivectors
A = 2.0*rand(N,d) - 1.0;
b = 2.0*rand(1,d) - 1.0;

fprintf('Array size: %d\n',N);

% vectorized multiplication
tic;
CA.product(A,b);
vector_time = toc;

% serial multiplication
tic;
for i = 1:N
    a = A(i,:);
    CA.product(a,b);
end
serial_time = toc;

fprintf('Performance factor: %g\n',serial_time/vector_time);
end

```

Appendix D Listing of geometric_algebra_test.m

```
%GEOMETRIC_ALGEBRA_TEST(N,tol)
%
%Unit tests of application of geometric algebra to projective geometry.
%
%Copyright (C) 2014 Defence Science and Technology Organisation
%
%Created by Leonid K. Antanovskii
function geometric_algebra_test(N,tol)
fprintf('Testing geometric algebra\n');
if nargin < 2
    rng('default');
    tol = 1.0e-13;
    if nargin < 1
        N = 100000;
    end
end

test_projective_space2D(tol,N);
test_projective_space3D(tol,N);
test_pappus_theorem(tol,N);
test_desargues_theorem(tol,N);
test_plucker_coordinates(tol,N);
test_epipolar_geometry(tol,N);
test_fundamental_map(tol,N);
test_point_reconstruction(tol,N);
test_line_reconstruction(tol,N);

end

%=====
function test_projective_space2D(tol,N)
fprintf('Incidence test in projective plane\n');

CA = clifford_algebra(3);

% original points
X1 = CA.set_vector(2.0*rand(N,3) - 1.0);
X2 = CA.set_vector(2.0*rand(N,3) - 1.0);
X3 = CA.set_vector(2.0*rand(N,3) - 1.0);

% lines passing through point pairs
L12 = CA.product_p(X1,X2);
L13 = CA.product_p(X1,X3);
L23 = CA.product_p(X2,X3);

% intersection points
Y1 = CA.product_r(L12,L13);
Y2 = CA.product_r(L12,L23);
Y3 = CA.product_r(L13,L23);

% comparison
q = CA.product_p(X1,Y1);
assert(all(q(:) < tol));
q = CA.product_p(X2,Y2);
assert(all(q(:) < tol));
q = CA.product_p(X3,Y3);
assert(all(q(:) < tol));
end

%=====
function test_projective_space3D(tol,N)
fprintf('Incidence test in projective space\n');

CA = clifford_algebra(4);
```

```

% original points
X1 = CA.set_vector(2.0*rand(N,4) - 1.0);
X2 = CA.set_vector(2.0*rand(N,4) - 1.0);
X3 = CA.set_vector(2.0*rand(N,4) - 1.0);
X4 = CA.set_vector(2.0*rand(N,4) - 1.0);

% planes passing through point triplets
P123 = CA.product_p(CA.product_p(X1,X2),X3);
P124 = CA.product_p(CA.product_p(X1,X2),X4);
P134 = CA.product_p(CA.product_p(X1,X3),X4);
P234 = CA.product_p(CA.product_p(X2,X3),X4);

% intersection points
Y1 = CA.product_r(CA.product_r(P123,P124),P134);
Y2 = CA.product_r(CA.product_r(P123,P124),P234);
Y3 = CA.product_r(CA.product_r(P123,P134),P234);
Y4 = CA.product_r(CA.product_r(P124,P134),P234);

% comparison
q = CA.product_p(X1,Y1);
assert(all(q(:) < tol));
q = CA.product_p(X2,Y2);
assert(all(q(:) < tol));
q = CA.product_p(X3,Y3);
assert(all(q(:) < tol));
q = CA.product_p(X4,Y4);
assert(all(q(:) < tol));
end

%=====
function test_pappus_theorem(tol,N)
fprintf('Pappus'' theorem test\n');

CA = clifford_algebra(3);

% first triplet of collinear points
A1 = CA.set_vector(2.0*rand(N,3) - 1.0);
B1 = CA.set_vector(2.0*rand(N,3) - 1.0);
C1 = rand*A1 + rand*B1;

% check collinearity
q = CA.product_p(CA.product_p(A1,B1),C1);
assert(all(q(:) < tol));

% second triplet of collinear points
A2 = CA.set_vector(2.0*rand(N,3) - 1.0);
B2 = CA.set_vector(2.0*rand(N,3) - 1.0);
C2 = rand*A2 + rand*B2;

% check collinearity
q = CA.product_p(CA.product_p(A2,B2),C2);
assert(all(q(:) < tol));

% intersection points
A = CA.product_r(CA.product_p(B1,C2),CA.product_p(B2,C1));
B = CA.product_r(CA.product_p(C1,A2),CA.product_p(C2,A1));
C = CA.product_r(CA.product_p(A1,B2),CA.product_p(A2,B1));

% check collinearity of the intersection points
q = CA.product_p(CA.product_p(A,B),C);
assert(all(q(:) < tol));
end

%=====
function test_desargues_theorem(tol,N)
fprintf('Desargues'' theorem test\n');

```

```

CA = clifford_algebra(3);

% triangle vertices in projective plane
A1 = CA.set_vector(2.0*rand(N,3) - 1.0);
B1 = CA.set_vector(2.0*rand(N,3) - 1.0);
C1 = CA.set_vector(2.0*rand(N,3) - 1.0);
A2 = CA.set_vector(2.0*rand(N,3) - 1.0);
B2 = CA.set_vector(2.0*rand(N,3) - 1.0);
C2 = CA.set_vector(2.0*rand(N,3) - 1.0);

% intersection points of triangle sides
A = CA.product_r(CA.product_p(B1,C1),CA.product_p(B2,C2));
B = CA.product_r(CA.product_p(C1,A1),CA.product_p(C2,A2));
C = CA.product_r(CA.product_p(A1,B1),CA.product_p(A2,B2));

% test for collinearity of points
p = CA.product_p(CA.product_p(A,B),C);

% lines through triangle vertices
La = CA.product_p(A1,A2);
Lb = CA.product_p(B1,B2);
Lc = CA.product_p(C1,C2);

% test for concurrency of lines
q = CA.product_r(CA.product_r(La,Lb),Lc);

% non-degenerate transformations
p = CA.dual(p);
i1 = CA.product_p(CA.product_p(A1,B1),C1);
i2 = CA.product_p(CA.product_p(A2,B2),C2);
s = -CA.product(i1,i2);
q = CA.product(s,q);

% (p = 0 <=> q = 0) follows from p = q
assert(norm(p - q,inf) < tol);
end

%=====
function test_plucker_coordinates(tol,N)
fprintf('Plucker coordinates test\n');

CA = clifford_algebra(4);

% random points
X1 = CA.set_vector(2.0*rand(N,4) - 1.0);
X2 = CA.set_vector(2.0*rand(N,4) - 1.0);

% lines through points
L = CA.product_p(X1,X2);

% check Klein quadric
q = klein_quadric(L);
assert(all(q < tol));

% random planes
P1 = CA.dual(X1);
P2 = CA.dual(X2);

% lines through planes
L = CA.product_r(P1,P2);

% check Klein quadric
q = klein_quadric(L);
assert(all(q < tol));
end

```

```

%=====
function test_epipolar_geometry(tol,N)
fprintf('Epipolar geometry test\n');

CA = clifford_algebra(4);

% first camera centre and projection plane
C1 = CA.set_vector(2.0*rand(1,4) - 1.0);
P1 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% second camera centre and projection plane
C2 = CA.set_vector(2.0*rand(1,4) - 1.0);
P2 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% base line and epipoles
B = CA.product_p(C1,C2);
O1 = CA.product_r(B,P1);
O2 = CA.product_r(B,P2);

% random points
X = CA.set_vector(2.0*rand(N,4) - 1.0);

% planes through the baseline and the points
P = CA.product_p(B,X);

% projection rays
L1 = CA.product_p(C1,X);
L2 = CA.product_p(C2,X);

% projection points
X1 = CA.product_r(P1,L1);
X2 = CA.product_r(P2,L2);

% epipolar lines
E1 = CA.product_r(P,P1);
E2 = CA.product_r(P,P2);

% check incidence
q = CA.product_p(E1,X1);
assert(all(q(:) < tol));
q = CA.product_p(E2,X2);
assert(all(q(:) < tol));
q = CA.product_p(E1,O1);
assert(all(q(:) < tol));
q = CA.product_p(E2,O2);
assert(all(q(:) < tol));
end

%=====
function test_fundamental_map(tol,N)
fprintf('Fundamental map test\n');

CA = clifford_algebra(4);

% first camera centre and projection plane
C1 = CA.set_vector(2.0*rand(1,4) - 1.0);
P1 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% second camera centre and projection plane
C2 = CA.set_vector(2.0*rand(1,4) - 1.0);
P2 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% base line and epipoles
B = CA.product_p(C1,C2);
O1 = CA.product_r(B,P1);
O2 = CA.product_r(B,P2);

```

```

% planes through the baseline and random points
P = CA.product_p(B,CA.set_vector(2.0*rand(N,4) - 1.0));

% epipolar lines induced by the planes
E1 = CA.product_r(P,P1);
E2 = CA.product_r(P,P2);

% check incidence with the epipoles
assert(norm(CA.product_p(O1,E1),inf) < tol);
assert(norm(CA.product_p(O2,E2),inf) < tol);

% action of the fundamental map
F1 = CA.product_r(CA.product_p(C2,E2),P1); % E2 -> F1
F2 = CA.product_r(CA.product_p(C1,E1),P2); % E1 -> F2

% check incidence with the epipoles
assert(norm(CA.product_p(O1,F1),inf) < tol);
assert(norm(CA.product_p(O2,F2),inf) < tol);

% check line equalities
for i = 1:N
    A = [
        line_matrix(E1(i,:))
        line_matrix(F1(i,:))
    ];
    assert(rank(A,tol) == 2);
end
for i = 1:N
    A = [
        line_matrix(E2(i,:))
        line_matrix(F2(i,:))
    ];
    assert(rank(A,tol) == 2);
end
end

%=====
function test_point_reconstruction(tol,N)
fprintf('Point reconstruction test\n');

CA = clifford_algebra(4);

% first camera centre and projection plane
C1 = CA.set_vector(2.0*rand(1,4) - 1.0);
P1 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% second camera centre and projection plane
C2 = CA.set_vector(2.0*rand(1,4) - 1.0);
P2 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% random points to project and recover
X = CA.set_vector(2.0*rand(N,4) - 1.0);

% projection rays
L1 = CA.product_p(C1,X);
L2 = CA.product_p(C2,X);

% projection points
X1 = CA.product_r(P1,L1);
X2 = CA.product_r(P2,L2);

% check inclusion using both outer products
assert(norm(CA.product_p(X1,P1),inf) < tol);
assert(norm(CA.product_p(X2,P2),inf) < tol);
assert(norm(CA.product_r(X1,P1),inf) < tol);
assert(norm(CA.product_r(X2,P2),inf) < tol);

```

```

% reconstructed projection rays
L1 = CA.product_p(C1,X1);
L2 = CA.product_p(C2,X2);

% check signed crossing values
q = CA.product_p(L1,L2);
assert(all(q(:) < tol));

% check incidence with ray 1
q = CA.product_p(L1,X);
assert(all(q(:) < tol));

% check incidence with ray 2
q = CA.product_p(L2,X);
assert(all(q(:) < tol));

% line intersection points
v = zeros(N,4);
for i = 1:N
    A = [
        line_matrix(L1(i,:))
        line_matrix(L2(i,:))
    ];
    [~,D,V] = svd(A,0);
    assert(abs(D(4,4)) < tol);
    v(i,:) = V(:,4)';
end
Y = CA.set_vector(v);

% check recovered triangulation points
q = CA.product_p(X,Y);
assert(all(q(:) < tol));
end

%=====
function test_line_reconstruction(tol,N)
fprintf('Line reconstruction test\n');

CA = clifford_algebra(4);

% first camera centre and projection plane
C1 = CA.set_vector(2.0*rand(1,4) - 1.0);
P1 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% second camera centre and projection plane
C2 = CA.set_vector(2.0*rand(1,4) - 1.0);
P2 = CA.dual(CA.set_vector(2.0*rand(1,4) - 1.0));

% random lines to project and recover
L = CA.product_p(...
    CA.set_vector(2.0*rand(N,4) - 1.0),...
    CA.set_vector(2.0*rand(N,4) - 1.0));

% projected lines
L1 = CA.product_r(P1,CA.product_p(C1,L));
L2 = CA.product_r(P2,CA.product_p(C2,L));

% check inclusion
assert(norm(CA.product_r(L1,P1),inf) < tol);
assert(norm(CA.product_r(L2,P2),inf) < tol);

% reconstructed lines
L0 = CA.product_r(CA.product_p(C1,L1),CA.product_p(C2,L2));

% check line equalities
for i = 1:N
    A = [

```

```

        line_matrix(L(i,:))
        line_matrix(L0(i,:))
    ];
    assert(rank(A,tol) == 2);
end
end

%=====
function q = klein_quadric(L)
l12 = L(:,6);
l13 = L(:,7);
l14 = L(:,8);
l23 = L(:,9);
l24 = L(:,10);
l34 = L(:,11);
q = l12.*l34 - l13.*l24 + l14.*l23;
end

%=====
function A = line_matrix(L)
A = [
    L(9),-L(7),L(6),0.0
    L(10),-L(8),0.0,L(6)
    L(11),0.0,-L(8),L(7)
    0.0,L(11),-L(10),L(9)
];
end

```

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA				1. CAVEAT/PRIVACY MARKING	
2. TITLE Implementation of Geometric Algebra in MATLAB® with Applications			3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)		
4. AUTHORS Leonid K. Antanovskii			5. CORPORATE AUTHOR Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia 5111, Australia		
6a. DSTO NUMBER DSTO-TR-3021		6b. AR NUMBER AR 016-076		6c. TYPE OF REPORT Technical Report	
7. DOCUMENT DATE September, 2014					
8. FILE NUMBER 2014/1169256/1	9. TASK NUMBER AIR07/213	10. TASK SPONSOR RAAF Air Combat Group	11. No. OF PAGES 48		12. No. OF REFS 10
13. URL OF ELECTRONIC VERSION http://www.dsto.defence.gov.au/ publications/scientific.php			14. RELEASE AUTHORITY Chief, Weapons and Combat Systems Division		
15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT <i>Approved for Public Release</i> OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111					
16. DELIBERATE ANNOUNCEMENT No Limitations					
17. CITATION IN OTHER DOCUMENTS No Limitations					
18. DSTO RESEARCH LIBRARY THESAURUS Science Mathematics Algebra Clifford Algebra Geometry Projective Geometry Algorithms Object Reconstruction					
19. ABSTRACT <i>Geometric Algebra</i> is the most appropriate unifying mathematical language to describe diverse problems in mathematics, physics, engineering and computer science. In combination with <i>Projective Geometry</i> it provides an efficient framework for computer vision and robotics, where image processing and recognition play the central rôle. This document addresses a gentle introduction to <i>Geometric Algebra</i> followed by its implementation in MATLAB. The developed fully vectorized code is thoroughly tested. Several applications are presented in the form of unit tests, amongst which are some basic algorithms for the reconstruction of a three-dimensional structure from two-dimensional images.					